2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering

Tackling Data Inefficiency: Compressing the Bitcoin Blockchain

Alexander Marsalek
A-SIT
Graz, Austria
Alexander.Marsalek@a-sit.at

Thomas Zefferer
A-SIT Plus GmbH
Vienna, Austria
Thomas.Zefferer@a-sit.at

Edona Fasllija
Graz University of Technology
Graz, Austria
Edona.Fasllija@iaik.tugraz.at

Dominik Ziegler
Know-Center GmbH
Graz, Austria
Dominik.Ziegler@tugraz.at

***Abstract*—In blockchain-based solutions, the amount of data stored in the blockchain increases steadily. Considerable amounts of data accordingly need to be downloaded and stored at decentralized nodes to carry out meaningful validations. For the Bitcoin blockchain, approximately 197 GB must currently be downloaded and processed by any node aiming to fully verify the correctness of stored transactions. This represents an emerging problem with respect to resource-constrained thin clients that lack the required download or storage capacities, but with which meaningful validations still need to be carried out. This problem will gain even more relevance in the future, as blockchain technology is applied to new domains such as the Internet of Things (IoT) that necessitate the use of thin clients. We address this emerging problem by proposing a novel compressible blockchain architecture. In our proposal, we use a snapshot-based approach, to create snapshots of the blockchain at regular intervals and store them in blocks. These blocks are chained, forming a second blockchain that is linked to the primary chain. In this way, the amount of data that needs to be downloaded and stored by decentralized nodes is reduced considerably, while the full validation of the entire blockchain still remains feasible. In this paper, we describe the proposed compressible blockchain architecture in detail. We conducted evaluations that demonstrate the feasibility of the proposed solution and show its advantages over related approaches introduced in the literature. Overall, our design can be used to reduce the size of the blockchain by up to 93%, facilitating secure blockchain-based applications even on resource-constrained thin clients such as IoT devices.**

***Index Terms*—compressible blockchain, UTXO, resource constrained devices, blockchain size, thin client, snapshot**

I. Introduction

Blockchain technology has grown in popularity during the past few years. This popularity has arisen mainly due to the hype around cryptocurrencies, for which blockchain technology acts as a technical backbone. However, during recent years, several other use cases have been expected to show a great potential for the use of blockchain technology. Although blockchain technology has not been as disruptive as predictions indicated up until now, many still expect that this technology will be used with increasing frequency in the future. Its advantages – also for use cases other than cryptocurrencies – are apparent. Blockchain technology reliably protects the immutability of stored data and renders central trusted parties unnecessary. Hence, it enables the development of secure and trustworthy decentralized applications. Possible fields of application include IoT [1], finance [2], energy [3], supply-chain management [4], and healthcare [5].

However, blockchain technology also is associated with certain disadvantages that threaten to hamper its broad application. One of these disadvantages is the continuously growing size of a blockchain. For instance, the size of the popular Bitcoin blockchain was 876 MB in the first quarter of 2012. In the first quarter of 2019, only seven years later, the size of the Bitcoin blockchain had grown to 197 GB [6]. To achieve the highest level of security and confidence, blockchain users have been advised to download and store a full copy of the blockchain locally. Only by doing so can the respective user carry out all validations that are required to have full confidence in the correctness of the stored data.

In general, requesting all users to store a full copy of the entire blockchain locally does not appear to be an ideal or efficient solution. However, this is the – generally accepted – price that must be paid to avoid the use of a trusted central party. Moreover, storing even several hundreds of GB of data is usually not a problem on modern desktop computers or laptops. However, compatibility with alternative client devices will become a requirement, as blockchain technology is about to make its way to new use cases and application scenarios. It is highly likely that some of these alternative devices will be more resource-constrained than desktop computers or laptops. For instance, storing a full copy of the Bitcoin blockchain can be a challenge even on current smartphones. Envisioning the use of even more resource-constrained devices, like the IoT, clearly indicates that the continuously growing size of blockchains can present severe practical problems.

An obvious solution to this problem appears to be avoiding the creation of full local copies. Indeed, it is possible to store only header information of the blocks instead of full copies of the blocks. Even with this reduced amount of information, the blockchain can still be validated on the client device. However, more sophisticated validations, such as validating new transactions or determining private balances, become infeasible without the use of trusted third parties. In use cases where security is critical, this can be an unacceptable restriction in terms of security and confidence.

In this paper, we address the problem of steadily growing blockchain sizes by introducing a compressible blockchain architecture. The proposed solution avoids the necessity of storing full copies of the blockchain on a client device, while maintaining the possibility of carrying out all relevant validations. At a glance, this is achieved by creating and

2324-9013/19/$31.00 ©2019 IEEE
DOI 10.1109/TrustCom/BigDataSE.2019.00089

storing cryptographically protected snapshots of the blockchain at regular time intervals. The proposed solution can be used to store the created snapshots in blocks that form a second blockchain that is linked to the original chain. By negating the need to store the complete, original blockchain, the proposed solution allows the application of blockchain technology on resource-constrained devices. In this way, new application scenarios become realizable, which require both the use of resource-constrained devices and compliance with strict security requirements.

A. Contributions

The work presented in this paper contributes to solving problems that are caused by the continuous growth of blockchain sizes. In particular, our contributions are three-fold:

Compressible Blockchain We propose and introduce a novel blockchain architecture that does not necessitate the creation of a full copy of the blockchain on each client device, but still enables all relevant validations to be carried out on these devices. The proposed blockchain architecture is especially intended for use on resource-constrained devices, but also enables a more efficient use of blockchain technology on classical devices like desktop computers or laptops.

Implementation We demonstrate the feasibility of the proposed blockchain architecture by means of a Java-based prototype implementation. The prototype demonstrates that all proposed concepts can be practically applied and are compatible with current blockchain implementations such as the Bitcoin blockchain.

Evaluation We evaluated our proposed solution using the developed prototype and data exported from the Bitcoin blockchain. The outcomes of the evaluations conducted show that this solution can be applied to reduce the amount of data that needs to be stored and downloaded by up to 93% compared to that stored and downloaded by a conventional Bitcoin node.

II. Background

Blockchain technology was developed to create a distributed trust ecosystem with no central authority interference by blending cryptographic mechanisms and P2P network exchanges. A public blockchain stores transaction data (i.e., transfer of ownership data) in a tamper-proof, append-only distributed ledger that consists of securely linked (chained) blocks.

Bitcoin [7], the first blockchain system proposed, enables its users to perform payments in that it allows them to execute a scripting language, and transfer BTC (Bitcoins) from transaction input(s) to transaction output(s). Miners include new valid transactions in candidate blocks by solving a probabilistically difficult proof-of-work puzzle. The new block is then broadcast to the other nodes where it is verified and included in the local copies of the blockchain.

This verification process mandates that all users or clients of the Bitcoin keep a local copy of the entire blockchain, including all transactions. Given the ever-growing size of the Bitcoin blockchain, this clearly poses a significant hindrance for lightweight clients, such as mobile or IoT devices. Nakamoto [7] addressed the necessity of reducing the storage overhead for lightweight clients in the original paper by proposing the *Simplified Payment Verification (SPV)* protocol. This protocol distinguishes between *full* and *lightweight* nodes. SPV enables the latter to verify their transactions without having to locally store the entire transaction history (around 197 GB at time of writing [6]), but instead only the block headers (each 80 bytes), of the longest chain. However, this lightweight verification process that is conducted by thin or lightweight clients is incomplete and limited as only block validity checks can be performed. The information contained in the block headers is sufficient to verify the valid structure of the blockchain at the lightweight nodes. The Merkel tree design enables thin clients to verify that the transactions they received have been included in the blockchain. Nevertheless, no insight is provided on whether a past transaction already exists in a chain. Hence, these clients cannot be used to perform transaction validity checks, and a vulnerability against double-spending attacks is introduced.

On the other hand, the entire blockchain is stored locally at full nodes and, based on this, an additional structure called the Unspent Transaction Output Set (UTXO set) is maintained. The UTXO set structurally stores all of the unspent transaction outputs (coins). Full nodes can then protect the system from the aforementioned double-spending attacks, by making sure that all of the inputs from a new transactions are present in the UTXO set. Construction and maintenance of the UTXO set also requires extra effort, as the entire chain is needed to build and update the UTXO set every time a new block is included. This explains why maintaining a UTXO set can be impractical on resource-constrained thin nodes. SPV nodes commonly rely on full nodes and trust that they have performed a full check on a block of interest in order to overcome this vulnerability. In summary, the inherent properties of the blockchain architecture presents challenges for resource-constrained thin clients.

In particular, there is a trade-off between the capability to carry out the meaningful validations of the blockchain and the need for local data storage. The solution proposed in this paper addresses this issue.

III. Related Work

The need to extend the blockchain technology's default functionality by improving scalability-features has been recognized early on by academia. In this section, we survey the related work that has addressed the problem of continuously growing amounts of data that need to be downloaded and stored locally.

An approach taken to improve scalability called sharding [8] was introduced early on by by Zamfir. In a classical blockchain, every full node is expected to validate all transactions and blocks. This limits scalability, as all validations are performed redundantly by each node. The idea behind sharding is to split the blockchain and its state, especially the UTXO set, into various subsets. Nodes hold only one subset and thus do not need to download, store and process all blocks. In this way, the entire load is shared across the network, and the number of

transactions per second can be increased significantly. However, for decentralized systems like Bitcoin, sharding is still a hard and not yet solved issue [9]. Unsolved problems exist in the context of cross-shard communication, fraud detection, and superquadratic sharding [10]. In summary, sharding appears to be a concept with high potential, but has not yet been successfully applied to existing blockchain implementations like Bitcoin.

Another approach that has been proposed in the literature is Segregated witness (segwit) [11]. This approach solves the scaling issue posed by blockchains by separating scripts and signatures from transactions and storing them in a separate data repository. This reduces the required space, as transaction signatures permanently occupy about 60% of the space required by the blockchain, but are only needed by fully-validating nodes and are only required once at the time of block verification. This approach has space-saving potential with regard to thin clients, which typically do not verify transaction signatures but still have to download them in classical blockchain solutions. Even full nodes that sync historical data do not validate all signatures. Instead, they typically use a checkpoint mechanism [12]. The segregated witness approach proposes to remove the signatures from the transactions and put them into a second tree. This makes it optional to propagate and store the signature. Critics claim that segregated witness increases the amount of network traffic and data storage by 400% but only gains a transactional throughput of 150% [13]. Overall, the approach pursues similar goals as our proposal. However, our approach is more efficient in that the required storage space can be reduced. For instance, applying the concept of segregated witness reduces the Bitcoin blockchain to approximately 118GB. In comparison, our proposal requires 3 GB to 13 GB (depending on chosen parameters) for the Bitcoin blockchain only.

Bruce [14] proposes a mini-blockchain scheme that can be used to solve the scalability problem. This scheme eliminates the need to store old transactions by unlinking them. Thus, the full blockchain is no longer required. All transactions older than a defined threshold can be removed. Old blocks are not completely deleted, as block headers are maintained. A prototype called Cryptonite [15] has been developed that is based on the Bitcoin code. An additional *account tree* assures that account balances can be calculated, even though old blocks are removed. This tree stores all non-empty addresses with its value and, thus, can be thought of as a balance sheet. To prune the *account tree*, an account maintenance fee is proposed. Thus, low-value accounts will reach a zero balance after some time and can then be removed from the balance sheet to save space. The work proposed by Bruce resembles our proposal in several respects. However, in our solution, the entire history is still available if needed, while thin clients do not need to verify all old blocks to achieve a comparable security level as full nodes.

Ehmke *et al.* [16] propose an approach based on concepts employed by Ethereum. To reduce the burden on nodes, the authors propose to keep the full state of the system in the current block. As a result, nodes can validate transactions without downloading and storing the full chain. The proposed approach introduces a dedicated proof that provides evidence that an address owns a certain property (proof-of-property). Relying on these proofs, validating nodes can verify transactions without knowing the balance of every address. In contrast to the solution proposed in our paper, the approach introduced by Ehmke *et al.* does not work with blockchain-based solutions that encode the state implicitly, like Bitcoin and most other cryptocurrencies.

CHAINIAC [17] is a decentralized approach proposed for transparent and verifiable software-updates. The authors introduce the concept of skipchains, a data structure that allows outdated nodes to efficiently verify updates or signing keys. To validate a target block, clients need to download only a logarithmic number of additional blocks. Thus, the approach is also suitable for resource-constrained thin clients, which cannot handle the full chain. Furthermore, the authors show that clients who have already cached a logarithmic number of blocks, including respective reference points, can verify the state without additional overhead. The ultimate goal of CHAINIAC is to provide a generic secure software framework. As a result, the authors do not focus on cryptocurrencies and do not take into account their specific characteristics. This distinguishes this approach from our solution, which is explicitly proposed to reduce the storage and download amount needed for cryptocurrencies like Bitcoin.

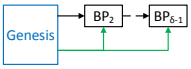
An obivious approach that can be taken to reduce data required for validations is to store the Unspent Transaction Output Set (UTXO set). A respective feature has been requested for cryptocurrencies like Bitcoin several times[1][2]. Some proposals request the provision of a signed UTXO set that can be imported to minimize the synchronization time. However, this raises several unsolved issues in a decentralized system that are related to trust relationships. To overcome such issues, we propose regularly creating a UTXO set and including its hash into new blocks. Thus, every block appended to the chain validates the accuracy of the claimed UTXO set. Details of how our approach can be used to store the UTXO set and all other concepts employed are introduced in the following section.

IV. COMPRESSIBLE BLOCKCHAIN

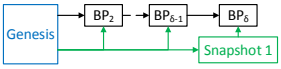
In this section we describe our proposed compressible blockchain solution. The main idea is to periodically create so-called snapshot blocks which contain the complete Unspent Transaction Output Set (UTXO set) and all block headers. The snapshot blocks build a second linked blockchain. Both chains start from the same trusted *Genesis* block. Figure 1 illustrates our approach. Figure 1a shows a blockchain with the second link pointing toward the genesis block. This link always points toward the last available snapshot block. Figure 1b shows the same chain after a snapshot block has been created. Now the second link points toward this snapshot block. The snapshot block itself is linked to its ancestor. Figure 1c shows the chain after some time has passed. In the mean-time, a new snapshot block has been created. The second chain is now recognizable.

[1] https://www.reddit.com/r/btc/comments/5uffmd/is_anyone_working_on_a_checkpoint_system_to/

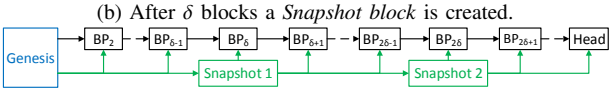
[2] https://bitcoincore.org/en/meetings/2016/03/10/\#initial-block-download-ibd-with-pre-generated-signed-utxos



(a) Standard blockchain with a second link pointing toward the genesis block.



(b) After δ blocks a *Snapshot block* is created.



(c) A compressible blockchain consisting of two linked chains, a primary chain and a *Snapshot* chain.

Fig. 1: Every δ blocks a new *Snapshot block* is created. The *Snapshot* blocks form a second linked chain.

For the remainder of this paper, we will refer to the first chain as the primary chain and to the second chain as the snapshot chain.

By using a deterministic algorithm, we can ensure that all nodes will end up with the same snapshot block. This means that the miners do not need to load it, instead they can always build it themselves. The extra effort invested to build this snapshot block is rather small, because miners need to know the current Unspent Transaction Output (UTXO) set in any case to verify received transactions and create valid new (standard) blocks. Furthermore, it is not necessary to store the complete second chain; the last few blocks are sufficient.

Next, we introduce the notation used in the remainder of the paper.

A. Notation

To introduce our proposed solution in detail, we extend the notation introduced by Garay *et al.*[18]. Accordingly, we refer to a block as any triple of the form

$$B = \langle \mathsf{s}, \mathsf{x}, \mathsf{ctr} \rangle \ ,$$

where $\mathsf{s} \in \{0,1\}^\kappa$, $\mathsf{x} \in \{0,1\}^*$ and $\mathsf{ctr} \in \mathbb{N}$. In this triple, s denotes the state (i.e., the hash value) of the previous block, x denotes the block's data, and ctr denotes the proof of work of the block. A block is considered valid, iff

$$validateBlock^D(B) := H(\mathsf{ctr}, G(\mathsf{s}, \mathsf{x})) < D$$

where $H : \{0,1\}^* \to \{0,1\}^\kappa$ and $G : \{0,1\}^* \to \{0,1\}^\kappa$ represent cryptographic hash functions. Furthermore, $D \in \mathbb{N}$ denotes the current difficulty of the cryptographic puzzle that needs to be solved to provide a correct proof of work.

Throughout this paper, we refer to a sequence of linked blocks as a chain C. New blocks are added to the right end of the chain. Accordingly, the first (and oldest) block is the leftmost block in the chain. The rightmost block (i.e., the newest block) is called the head of the chain, denoted by $head(C)$. Extending a chain C with $head(C) := \langle \mathsf{s}, \mathsf{x}, \mathsf{ctr} \rangle$ by attaching (to the right end), a valid block $B' := \langle \mathsf{s}', \mathsf{x}', \mathsf{ctr}' \rangle$ where $\mathsf{s}' := H(\mathsf{ctr}, G(\mathsf{s}, \mathsf{x}))$ yields a longer chain C'. We refer to the number of blocks in chain C as the length of chain $len(C)$. Finally, B_i denotes the i-th block in the chain C. Accordingly, B_1 is the genesis (first) block of the chain C and $B_n = head(C) \iff len(C) = n$. The same holds true for the fields of a block, e.g., s_i references the value of the s field in the i-th block. Note that the value of s_i changes if a block is added before the i-th index in the chain.

Algorithm 1 extendBlockchain

Input: Chain C consisting of $C\mathcal{P} = (\mathcal{BP}_1, ..., \mathcal{BP}_n)$ of length n, chain $C\mathcal{S} = (\mathcal{BS}_1, ..., \mathcal{BS}_m)$, the block data xp, the snapshot interval δ and the current UTXO set U.
Output: $\{\langle \mathcal{BP}, \mathcal{BS} \rangle\}$

```
1: if n + 1 mod δ = 0 then
2:     #Build a new snapshot block
3:     BS_{m+1} ← ⟨H(BS_m), sort(U), H(BP_n)⟩
4:     BP := ⟨H(BP_n), xp, ctrp', cp_{m+1}⟩ #ctrp' is determined by mining
5:     return ⟨BP, BS_{m+1}⟩
6: else
7:     #Snapshot interval not reached, create only a normal block
8:     BP := ⟨H(BP_n), xp, ctrp', cp_m⟩   #ctrp' is determined by mining
9:     return ⟨BP, null⟩
```

B. Protocol

The proposed compressible blockchain C consists of two linked blockchains, the *Primary Chain* $C\mathcal{P}$ and the *Snapshot Chain* $C\mathcal{S}$, both stemming from a common *Genesis Block*. Blocks in the *Primary Chain* $C\mathcal{P}$ have the form

$$\mathcal{BP} := \langle \mathsf{sp}, \mathsf{xp}, \mathsf{ctrp}, \mathsf{cp} \rangle \ ,$$

where $\mathsf{sp} \in \{0,1\}^\kappa$, $\mathsf{xp} \in \{0,1\}^*$, $\mathsf{ctrp} \in \mathbb{N}$, and $\mathsf{cp} \in \{0,1\}^\kappa$. Here, sp is the state (i.e., hash) of the previous block in $C\mathcal{P}$, xp is the data in the block $\mathcal{BP}$, ctrp represents the proof of work, and $\mathsf{cp} \in \{0,1\}^\kappa$ is the state (i.e., hash) of the last snapshot block in $C\mathcal{S}$. Blocks in the *Snapshot Chain* $C\mathcal{S}$ have the form

$$\mathcal{BS} := \langle \mathsf{ss}, \mathsf{xs}, \mathsf{st} \rangle \ ,$$

where $\mathsf{ss} \in \{0,1\}^\kappa$, $\mathsf{xs} \in \{0,1\}^*$ and $\mathsf{st} \in \{0,1\}^\kappa$. Here, ss is the state (i.e. hash) of the previous block in $C\mathcal{S}$, xs is the data of the block $\mathcal{BS}$, and st is the hash of the last block in $C\mathcal{P}$ considered for the snapshot. xs stores the UTXO set. Depending on the use case, it is also possible to store all headers of blocks in $C\mathcal{P}$ in the field xs. We define the hash of blocks $\mathcal{BP}$ in the *Primary Chain* as

$$H(\mathcal{BP}) := H(\mathsf{ctrp}, G(\mathsf{sp}, \mathsf{xp}, \mathsf{cp})) \ .$$

Furthermore, we define the hash of blocks $\mathcal{BS}$ of the *Snapshot Chain* as

$$H(\mathcal{BS}) := H(G(\mathsf{ss}, \mathsf{xs}, \mathsf{st})) \ .$$

We define the snapshot interval as δ. $\delta \in \mathbb{N}$ defines the number of blocks after which a new snapshot block has to be created.

Adding a Block: A chain C, with $head(C\mathcal{P}) = \mathcal{BP}_n$ and $head(C\mathcal{S}) = \mathcal{BS}_m$ can be extended to a longer chain by appending a new block $\mathcal{BP} := \langle H(\mathcal{BP}_n), \mathsf{xp}', \mathsf{ctrp}', \mathsf{cp}' \rangle$ using Algorithm 1. If the second parameter (i.e. $\mathcal{BS}_{m+1}$) is not *null*, this block must be added to $C\mathcal{S}$. Note that if the snapshot interval has been reached, a snapshot block is built. Next we explain how the proposed blockchain is validated.

Algorithm 2 validateFullBlockchain

Input: Chain $\mathcal{C}$ consisting of $\mathcal{CP} = (\mathcal{BP}_1, ..., \mathcal{BP}_n)$ of length n, chain $\mathcal{CS} = (\mathcal{BS}_1, ..., \mathcal{BS}_m)$, the snapshot interval δ
Output: {0, 1}

$\mathcal{G} \leftarrow GenesisBlock$
if $\mathcal{BP}_1 \neq \mathcal{G} \vee \mathcal{BS}_1 \neq \mathcal{G}$ **then return** 0
$\mathsf{U} \leftarrow validate\mathcal{BP}(\mathcal{BP}_1, \{\})$ *#Initialize empty UTXO set*
#Call Algorithm 4
return $validateBlockchain(\mathcal{C}, \delta, U)$

Algorithm 3 validateThinClientBlockchain

Input: Chain $\mathcal{C}$ consisting of $\mathcal{CP} = (\mathcal{BP}_1, ..., \mathcal{BP}_n)$ of length n, chain $\mathcal{CS} = (\mathcal{BS}_1, ..., \mathcal{BS}_m)$, the snapshot interval δ
Output: {0, 1}

#Verify if both chains start with a trusted block
if $\mathcal{BP}_1 \neq \mathcal{BP}_{Trust}$ **then return** 0
if $\mathcal{BS}_1 \neq \mathcal{BS}_{Trust}$ **then return** 0
#Load UTXO set from $\mathcal{BS}_1$
$\langle \mathsf{U}, headers \rangle \leftarrow parse\mathsf{xs}(\mathsf{xs}_1)$
#Initialize UTXO set U
$\mathsf{U} \leftarrow validate\mathcal{BP}(\mathcal{BP}_1, U)$
#Verify that the headers build a valid chain starting from the Genesis Block $\mathcal{G}$
if $verifyHeaders(headers, \mathcal{G}) \neq valid$ **then**
  return 0
#Call Algorithm 4
return $validateBlockchain(\mathcal{C}, \delta, U)$

Full Chain Validation: Algorithm 2 is used by full nodes to validate a given compressible chain $\mathcal{C}$. It takes the chain $\mathcal{C}$ and the snapshot interval δ as input. The algorithm is used to verify that both chains start with the trusted *GenesisBlock*, initialize the UTXO set from the *GenesisBlock*, and call Algorithm 4.

Thin Client Validation: When synchronization is required on a new thin client it uses Algorithm 5 to load the chain $\mathcal{C}$. Algorithm 5 takes a confidence factor α as input, defining the minimum number of blocks in $\mathcal{CP}$ necessary, that confirm the validity of a snapshot block in order to trust it. After a thin client has received a chain, it can verify it using Algorithm 3. If the thin client receives different chains from the network, it uses the valid one with the highest combined proof-of-work. This can be determined by analyzing the block headers. Note that Algorithm 5 and 3 are separated for better understandability. For performance reasons, this should be avoided in real-world applications.

Practical Considerations: For real-world applications we recommend that the current UTXO set is not included in the snapshot block, but instead the UTXO set from two blocks ago. This levels the field; otherwise, the miner that finds the last block before the snapshot interval would have an unfair advantage. Including the UTXO set from two blocks ago allows all miners to prepare the Snapshot block (and the UTXO set) in advance. In this way, our proposed solution ascertains no miner is given an unfair advantage.

V. Evaluation

To test the proposed solution in practice, we carried out an evaluation comprising the following steps. First, we developed a prototype implementation of the proposed compressible blockchain architecture. Using this prototype, we benchmarked

Algorithm 4 validateBlockchain

Input: Chain $\mathcal{C}$ consisting of $\mathcal{CP} = (\mathcal{BP}_1, ..., \mathcal{BP}_n)$ of length n, chain $\mathcal{CS} = (\mathcal{BS}_1, ..., \mathcal{BS}_m)$, the snapshot interval δ and the initial UTXO set U.
Output: {0, 1}

$j \leftarrow 1$ *#Index of the last verified block in* $\mathcal{CS}$
#Hash value of the last verified block in $\mathcal{CS}$
$h\mathcal{BS} \leftarrow H(\mathcal{BS}_1)$
#Iterate of all blocks in $\mathcal{CP}$*, starting with* $\mathcal{BP}_2$
for $i = 2$ to n **do**
  #Verify data in $\mathcal{BP}_i$ *and update the UTXO set* U *like in Bitcoin*
  $\langle valid, \mathsf{U} \rangle \leftarrow validateDataAndUpdateUTXOSet(\mathsf{xp}_i, \mathsf{U})$
  #Check difficulty of block
  if $valid \neq 1 \vee H(\mathsf{ctrp}_i, G(\mathsf{sp}_i, \mathsf{xp}_i, \mathsf{st}_i)) \geq D$ **then**
    return 0
  #Validate chain linking
  if $\mathsf{sp}_i \neq H(\mathsf{ctrp}_{i-1}, G(\mathsf{sp}_{i-1}, \mathsf{xp}_{i-1}, \mathsf{st}_{i-1}))$ **then return** 0
  #Check if it is time for a new snapshot block
  if $i \mod \delta = 0$ **then**
    $\mathcal{BS} \leftarrow \langle h\mathcal{BS}, sort(\mathsf{U}), H(\mathcal{BP}_i) \rangle$
    #Verify that the snapshot block matches our expectation
    if $H(\mathcal{BS}) \neq H(\mathcal{BS}_{j+1})$ **then return** 0
    $h\mathcal{BS} \leftarrow H(\mathcal{BS})$
    $j \leftarrow j + 1$
  else
    if $\mathsf{cp}_i \neq h\mathcal{BS}$ **then**
      return 0
#Verify that all snapshot blocks have been processed.
if $m \neq j$ **then**
  return 0
else
  return 1

our approach on real data retrieved from the Bitcoin blockchain. We describe these two steps in detail in the following subsections. In addition, we discuss the possible security implications of the proposed approach.

A. Prototype Implementation

First, we developed a Java-based prototype implementation of our solution. In this way, we successfully demonstrated the feasibility of the proposed compressible blockchain architecture. We intentionally kept the prototype as simple as possible and placed a focus on the aspects we considered as most relevant with regard to our proposal. For example, the prototype implementation does not rely on a P2P network; it uses a "bootstrap" server. This server stores and forwards all transactions and blocks. Despite the applied simplifications, the developed prototype successfully demonstrates the feasibility of our proposal.

B. Benchmark

Next, we conducted a series of measurements to benchmark the storage-saving potential of the proposed solution in practice. We used the prototype implementation introduced in Section V-A and applied its snapshot functionality to a copy of the Bitcoin blockchain. Using real data from the Bitcoin blockchain assures the meaningfulness of the benchmarking results obtained.

To evaluate these results in more detail, we carried out the following steps to benchmark our proposed solution:

- We created a new Bitcoin full node and set the *bitcoind* daemon parameter "stopatheight" to value 10,000.

Algorithm 5 synchronizeThinClient

Input: Confidence factor α
Output: Chain $\mathcal{C}$

$\mathcal{CP} \leftarrow \{\}$
$\mathcal{CS} \leftarrow \{\}$
#Load head block of $\mathcal{CP}$ from network
$\mathcal{BP}_{Head} \leftarrow loadHead()$
#Add $\mathcal{BP}_{Head}$ to the beginning of $\mathcal{CP}$
$\mathcal{CP} \leftarrow \mathcal{BP}_{Head} \cup \mathcal{CP}$
#Load snapshot block $\mathcal{BS}_{Head}$ referenced in $\mathcal{BP}_{Head}$ by block field cp_{Head} from network
$\mathcal{BS}_{Head} \leftarrow loadBlock(\mathsf{cp}_{Head})$
#Add $\mathcal{BS}_{Head}$ to the beginning of $\mathcal{CS}$
$\mathcal{CS} \leftarrow \mathcal{BS}_{Head} \cup \mathcal{CS}$
$i \leftarrow 1$
repeat
  $prevBlockHash \leftarrow ss_1$
  $target \leftarrow \mathsf{st}_1$
  #Load all blocks between $\mathcal{BP}_{Head}$ (referenced by ss_1) and the block referenced by the first snapshot block via st_1
  repeat
    $\mathcal{BP}_x = loadBlock(prevBlockHash)$
    #Add block to the beginning of $\mathcal{CS}$
    $\mathcal{CP} \leftarrow \mathcal{BP}_x \cup \mathcal{CP}$
    #Increase the number of blocks confirming our snapshot block
    $i \leftarrow i + 1$
    #Retrieve the hash value of the previous block of $\mathcal{BP}_x$
    $prevBlockHash \leftarrow \mathsf{sp}_x$
    #Check if target hash value has been reached
  until $prevBlockHash \neq target$
  #old...
  for each $\mathcal{BP} \in [\mathsf{sp}_1, \mathsf{st}_1)$ **do**
    #Add block at the beginning
    $\mathcal{CP} \leftarrow \mathcal{BP} \cup \mathcal{CP}$
    $i \leftarrow i + 1$
  #Verify if at least α confirm the snapshot block
  if $i < \alpha$ **then**
    #Add block at beginning of $\mathcal{CS}$, thus changing st_1
    $\mathcal{BS} \leftarrow loadBlock(\mathsf{ss}_{Head})$
    $\mathcal{CS} \leftarrow \mathcal{BS} \cup \mathcal{CS}$
until $i \geq \alpha$
return $\mathcal{C}$

- We then started synchronizing the created node. Due to the set parameter "stopatheight"[3], the synchronization stopped after the chain had reached the given limit
- Next, we used the remote procedure call (RPC) "gettxoutsetinfo" to retrieve information on the Unspent Transaction Output Set (UTXO set) of the (partly) synchronized node.

We repeated the steps described above while increasing the value of parameter "stopatheight" by 10,000 with each repetition. We continued this process until the node was fully synchronized. In this way, we retrieved information on the UTXO set at 56 different points over the Bitcoin blockchain's lifetime. Finally, we used the RPC "getblock" to retrieve relevant information on all (approx. 565,500) blocks within the chain. All results retrieved are illustrated in the two subplots of Figure 2.

The two subplots of Figure 2 show the same data, but they were respectively plotted on linear (left subplot) and logarithmic (right subplot) scales for better visibility. Figure 2 shows that the overall size of the Bitcoin blockchain has not grown linearly. One reason for this becomes also apparent in Figure 2: Especially in the early days of Bitcoin, smaller blocks were added, as the number of Bitcoin transaction was still low. As the growing popularity of Bitcoin grew, this changed. Nowadays, the maximum block size is usually fully employed. Figure 2 also shows that the UTXO set size is significantly smaller as compared to the size of the complete blockchain. This finding supports our assumption that the download and storage capacity can be saved by focusing purely on the UTXO set. Note that all overhead (e.g., network, etc.) was removed from the dataset shown in Figure 2.

After this first data collection and analysis step, we applied the proposed compression feature provided by the developed prototype. The proposed mechanism can basically be adapted by modifying two parameters: First, the snapshot interval δ can be used to define the number of blocks, after which a new snapshot is created. Second, the confidence factor α can be used to define the minimum number of blocks that need to follow a snapshot, in order for the user to have sufficient trust in the accuracy of the snapshot.

Using the data collected previously, we determined the resulting (reduced) amount of data that needs to be downloaded and stored by a node for different values of δ and α. We ran experiments at snapshot intervals δ of 10,000, 20,000, 50,000, and 100,000 blocks, and with the confidence factor α set at 100, 500, and 1,000 blocks. Figure 3 illustrates the results obtained from the conducted test runs.

In Figure 3, each plot represents one test run with a specific combination of values for the two parameters δ and α. An examination of Figure 3 reveals that the amount of data that needs to be downloaded and stored locally grows as the size of the snapshot interval δ grows. Furthermore, Figure 3 illustrates the fact that the confidence factor α has a minor impact only.

Figure 4 provides more detailed information about a specific part of Figure 3. The provided close-up shows results obtained for the blocks 500,000 to 525,000. Note that the y-axis is cropped. As expected, Figure 4 shows that the confidence factor α has a small impact.

Finally, Figure 5 provides a comparison of the amount of data that must be downloaded by a new node for different blockchain heights. Figure 5 shows that even using large snapshot intervals δ improve the situation, while using a smaller δ yields even better improvements. Note that old snapshots do not need to be stored. If they are needed, old snapshots can always be recreated. New nodes will probably request only the last few snapshots, depending on their requirements regarding the confidence factor.

Overall, the outcome of the benchmarks conducted and results obtained demonstrate that our proposed solution can be applied to reduce the amount of data that needs to be downloaded and stored by conventional Bitcoin nodes by up to 93%.

C. Security Considerations

Security is an important aspect that must be evaluated when extending the feature set of blockchain technology. It needs to be assured that added functionality does not reduce the security. In the following section, we discuss the security implications of our proposed feature-set extension.

[3]https://github.com/bitcoin/bitcoin/blob/master/src/init.cpp Line 488

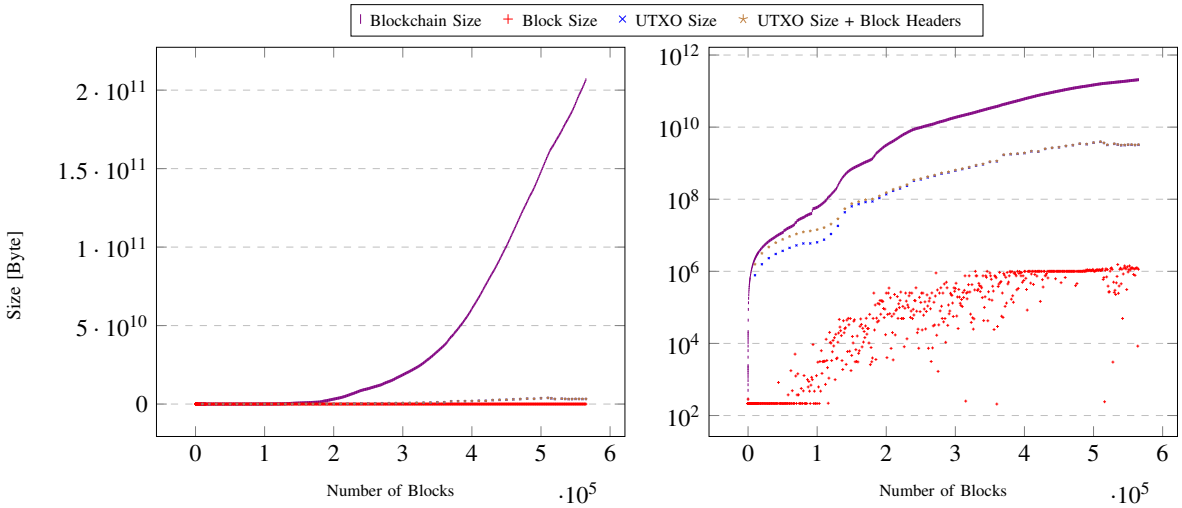


Fig. 2: A visualization of size of the Bitcoin blockchain, its blocks, the UTXO set and the UTXO set plus all previous block headers. For better perception on the left side with a linear y-axis and a logarithmic y-axis on the right side.

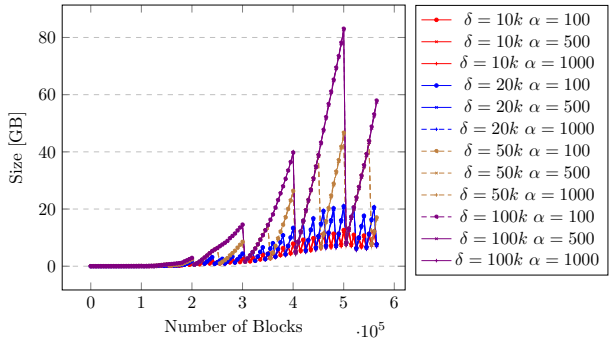


Fig. 3: This figure shows the data that must be downloaded to synchronize a new node for different blockchain lengths and snapshot intervals δ ranging from 10,000 blocks to 100,000 blocks. The second parameter in the legend is the confidence factor α.

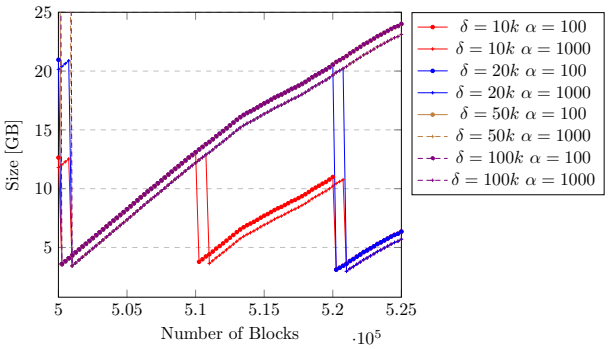


Fig. 4: This figure is a zoomed-in version of Figure 3, showing the negligible impact of the confidence factor α in terms of data to download or store. For clarity reasons we removed the $\alpha = 500$ plots.

Our proposal targets blockchain solutions that rely on a consensus algorithm based on proof-of-work. We accordingly assume that more than 50% of the network's computation power is held by nodes following the protocol. Furthermore we assume that our proposed additional features extend a secure blockchain system. Thus, it is sufficient to show that the proposed additional features do not weaken the overall security.

Obviously, the security of full nodes is not affected by our approach, as our proposal does not change the validation processes conducted by full nodes at all. Our approach only adds new rules, which enforce the periodic generation of a (valid) snapshot block that is referenced in the primary chain. For thin clients with limited resources, an SPV solution has been typically used so far, due to the lack of more powerful approaches. The proposed approach resembles that taken in

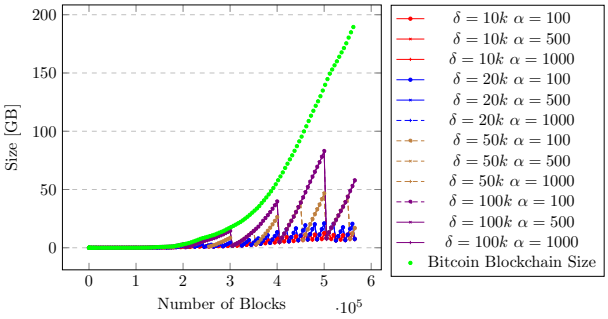


Fig. 5: This figure visualizes the difference of the data amount needed to synchronize our solution compared to the standard Bitcoin blockchain for different snapshot intervals and confidence factors.

SPV solutions, but it provides more functionality. Like SPV, our proposal enables thin clients to verify all block headers. Thus, our proposal reaches at least the same level of security as SPV solutions. However, once the thin client has been fully synchronized, our approach provides higher levels of security and privacy. As the complete UTXO set is known to the thin client, it can verify every unconfirmed transactions that is received. In contrast, simple SPV nodes have to trust that connected nodes do not provide them with invalid unconfirmed transactions. Furthermore, SPV nodes have to query full nodes for blocks that contain relevant transactions, which leads to a trade-off between resource saving and privacy.

In summary, our proposed solution provides the same level of security for full nodes as classical blockchain solutions, but it also provides better security for thin clients than SPV solutions. As SPV is regarded as sufficiently secure, the same holds for our proposal.

VI. Conclusions

The continuously growing size of blockchains can present problems, especially in use cases that rely on resource-constrained devices such as smartphones. In this paper, we have proposed and introduced a compressible blockchain architecture. Our proposal negates the need to store full copies of the blockchain on the client device, but still enables the use of this device to fully validate all relevant aspects of the blockchain. This distinguishes our solution from other approaches, which also can be taken to reduce the necessary amount of data that needs to be stored, but which only provide limited validation capabilities. Our evaluation shows that we can successfully compress the Bitcoin blockchain by up to 93% without weakening the security assumptions.

In future work, we plan to further improve our snapshot functionality. Instead of storing full snapshots that contain the entire blockchain, we will evaluate how to store incremental updates in the created snapshots, focusing on a creating a well balanced combination of full and incremental snapshots.

We believe that we can further improve our proposed compressible blockchain architecture by effectively combining full and incremental snapshot blocks and plan to implement and evaluate this architecture in detail.

The proposed compressible blockchain architecture already constitutes a clear improvement compared to classical blockchains in its current form. By negating the need to locally store full copies of the blockchain, our solution makes it possible to use blockchain technology on resource-constrained devices and thus makes this technology an attractive option for various new use cases.

References

[1] M. Singh, A. Singh, and S. Kim, "Blockchain: A game changer for securing iot data," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, IEEE, 2018, pp. 51–55.

[2] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, no. 9, 2017.

[3] N. Z. Aitzhan and D. Svetinovic, "Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams," *IEEE Transactions on Dependable and Secure Computing*, 2018. DOI: 10.1109/TDSC.2016.2616861.

[4] J.-A. Vorabutra, "Why blockchain is a game changer for supply chain management transparency," *Supply Chain 24*, vol. 7, 2016.

[5] M. Mettler, "Blockchain technology in healthcare: The revolution starts here," in *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, IEEE, 2016, pp. 1–3.

[6] Statista 2019, *Size of the bitcoin blockchain from 2010 to 2019, by quarter (in megabytes)*. URL: https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/.

[7] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*. URL: https://bitcoin.org/bitcoin.pdf.

[8] V. Zamfir, *Sharding the blockchain*. URL: https://diyhpl.us/wiki/transcripts/scalingbitcoin/sharding-the-blockchain/.

[9] P. Todd, *Why Scaling Bitcoin With Sharding Is Very Hard*. URL: https://petertodd.org/2015/why-scaling-bitcoin-with-sharding-is-very-hard.

[10] V. Buterin, *Sharding faq*. URL: https://github.com/ethereum/wiki/wiki/Sharding-FAQ.

[11] P. E. Lombrozo, J. Lau, and P. Wuille, *Bip141: Segregated witness (consensus layer)*, 2015. URL: https://en.bitcoin.it/wiki/BIP_0141.

[12] P. Wuille, "Segregated witness and its impact on scalability," in *SF Bitcoin Devs Seminar*, 2015.

[13] C. S. Wright, *The illusion of scale in segregated witness*, 2017. URL: http://nchain.com/app/uploads/2017/07/SegWit-and-the-illusion-of-scale.pdf.

[14] J. Bruce, *The mini-blockchain scheme*. URL: https://cryptonite.info/files/mbc-scheme-rev3.pdf.

[15] Mini-blockchain Project 2017, *Cryptonite*. URL: https://cryptonite.info/.

[16] C. Ehmke, F. Wessling, and C. M. Friedrich, "Proof-of-property: A Lightweight and Scalable Blockchain Protocol," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, ser. WETSEB '18, New York, NY, USA: ACM, 2018, pp. 48–51, ISBN: 978-1-4503-5726-5.

[17] K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, N. Gailly, L. Gasser, I. Khoffi, J. Cappos, and B. Ford, "CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1271–1287.

[18] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 281–310, ISBN: 978-3-662-46803-6.