CoinPrune: Shrinking Bitcoin's Blockchain Retrospectively

Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle

***Abstract*—Popular cryptocurrencies continue to face serious scalability issues due to their ever-growing blockchains. Thus, modern blockchain designs began to prune old blocks and rely on recent snapshots for their bootstrapping processes instead. Unfortunately, established systems are often considered incapable of adopting these improvements. In this work, we present *CoinPrune*, our block-pruning scheme with full Bitcoin compatibility, to revise this popular belief. CoinPrune bootstraps joining nodes via snapshots that are periodically created from Bitcoin's set of unspent transaction outputs (UTXO set). Our scheme establishes trust in these snapshots by relying on CoinPrune-supporting miners to *mutually reaffirm* a snapshot's correctness on the blockchain. This way, snapshots remain trustworthy even if adversaries attempt to tamper with them. Our scheme maintains its retrospective deployability by relying on positive feedback only, i.e., blocks containing invalid reaffirmations are not rejected, but invalid reaffirmations are outpaced by the benign ones created by an honest majority among CoinPrune-supporting miners. Already today, CoinPrune reduces the storage requirements for Bitcoin nodes by two orders of magnitude, as joining nodes need to fetch and process only 6 GiB instead of 271 GiB of data in our evaluation, reducing the synchronization time of powerful devices from currently 7 h to 51 min, with even larger potential drops for less powerful devices. CoinPrune is further aware of higher-level application data, i.e., it conserves otherwise pruned application data and allows nodes to obfuscate objectionable and potentially illegal blockchain content from their UTXO set and the snapshots they distribute.**

***Index Terms*—blockchain; block pruning; synchronization; bootstrapping; scalability; velvet fork; Bitcoin**

I. Introduction

KEY to the success of cryptocurrencies such as Bitcoin [1] is the blockchain, an immutable append-only ledger of financial transactions. Using the blockchain, all nodes can independently verify the transaction history, allowing mutually distrusting peers to establish consensus about the correctness of those transactions. This consensus also enables, for instance, audit systems [2]–[4], transparency overlays [5], [6], bootstrapping anonymity services [7], and smart contracts [8].

However, besides all benefits blockchain systems face serious scalability challenges, e.g., limited transaction throughput, high payment verification delays [9], and, most importantly, ever-growing blockchain sizes: Bitcoin's blockchain exhibits a size of 281 GiB with a recent average growth rate of 143 MiB/day as of Sep 29, 2020 [10]. Furthermore, the unintended utilization of Bitcoin's blockchain as a general-purpose data storage [11]–[13] has put a permanent burden onto the system and its users, as (a) such misuse typically bloats the set of unspent transaction outputs (UTXO set) with entries that are never spendable and (b) objectionable content can irrevocably be engraved into the blockchain and is subsequently distributed to all nodes [14]. Large blockchain sizes and the presence of objectionable blockchain content cause individual nodes to *prune* older blockchain data [15], i.e., older payment flows that have been superseded by newer ones, or *locally erase* UTXOs that hold unwanted content [16] at the cost of becoming dependent on other nodes for transaction validation.

While such decisions to prune or erase local blockchain data are rational from the users' perspective, they harm the overall network health: To root trust in Bitcoin's current state, new nodes need to obtain and revalidate *all* blockchain data, including all now-obsolete data, from independent sources. Addressing this inherent conflict of interests, alternative designs [17]–[20] proposed *snapshot-based synchronization*, where new nodes do not verify all blockchain data but instead rely on a recent snapshot of the blockchain's state. While these efforts solve scalability challenges for new blockchain systems, they do not readily carry over to existing and well-established blockchain systems, such as Bitcoin, where significant changes have to be adopted by a majority of nodes [21]. Consequently, there is a need to extend *existing* blockchain systems with pruning capabilities in a secure and trustworthy manner without the need for significant changes of the underlying protocol.

Our Contributions. We propose *CoinPrune*[1], a block-pruning scheme that is fully compatible with Bitcoin and can be adopted immediately by any subset of nodes without changing Bitcoin's consensus rules. At its core, CoinPrune provides a *trustworthy* and fully distributed snapshot-based bootstrapping process based on Bitcoin's UTXO set, which is significantly smaller than Bitcoin's full blockchain. Consequently, CoinPrune drastically unburdens the whole Bitcoin network as joining nodes require only the small snapshots for bootstrapping, and therefore established nodes can prune obsolete data *without* impeding the overall network health. In summary, we make the following contributions in this paper:

1) We *comprehensively survey* existing approaches to decrease the storage requirements and synchronization times of blockchain systems, concluding that they either are inefficient, insecure, or not deployable to existing systems.
2) We present CoinPrune, a snapshot-based pruning scheme that is fully compatible with Bitcoin. CoinPrune establishes

Roman Matzutt, Benedikt Kalde, Jan Pennekamp, and Klaus Wehrle are with the Chair of Communication and Distributed Systems, RWTH Aachen University, Aachen, Germany (e-mail: {lastname}@comsys.rwth-aachen.de).

Arthur Drichel is with the IT Security Research Group, RWTH Aachen University, Aachen, Germany (e-mail: drichel@itsec.rwth-aachen.de).

Martin Henze is with the Cyber Analysis & Defense Department, Fraunhofer FKIE, Wachtberg, Germany (e-mail: martin.henze@fkie.fraunhofer.de).

[1]Research prototype available at https://github.com/COMSYS/coinprune
Snapshots available at https://coinprune.comsys.rwth-aachen.de

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021).
Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

trust because CoinPrune-supporting miners *periodically and independently reaffirm* the snapshots' correctness by cryptographically tying the current snapshot to the blockchain. Thus, assuming a sufficient level of CoinPrune support, joining nodes may trust the honest network majority to verify snapshots before relying on them. As legacy nodes ignore rather than reject reaffirmations, CoinPrune can be deployed mid-operation via a velvet fork [22], [23].

3) CoinPrune further renders objectionable content in the UTXO set (and thus its snapshots) harmless by *obfuscating* values to make the content inaccessible without losing the corresponding transactions' verifiability. Simultaneously, CoinPrune preserves short chunks of non-financial data from `OP_RETURN` transaction outputs, which are essential for higher-level applications such as audit systems [2]–[4], in an additional *application data storage*.
4) Our evaluation shows that CoinPrune reduces disk space utilization for full nodes by 87 % while still enabling new nodes to synchronize. Pruning improves the synchronization performance drastically: Network traffic is reduced by 93 % and synchronization times drop from 7 h to 51 min on powerful devices with even larger drops on less powerful devices. Further, obfuscating snapshots keeps current snapshot sizes at 4.63 GiB, and retaining all `OP_RETURN` data together with relevant metadata requires 9.20 GiB.

A preliminary version of this paper appears in the proceedings of IFIP Networking 2020 [24]. We extend and improve upon our previous work in the following ways: First, we extend CoinPrune to *obfuscate almost all objectionable content* [12] to make such content inaccessible without hurting transaction verifiability. Second, we *extend our security discussion* by simulatively assessing an adversary's success chance to reaffirm an invalid snapshot. Third, we introduce an application data storage for the *preservation of application-level data*. Fourth, we *extend our discussion of related work* to also cover new approaches and work that does not primarily target synchronization scalability. Finally, we *updated our performance evaluation* with newer data from Bitcoin's blockchain.

Paper Structure. Section II provides the necessary background on Bitcoin. In Section III, we discuss the negative impacts of growing blockchains. We then extensively survey related work in Section IV and identify requirements for secure block-pruning schemes based on this survey in Section V. Section VI provides an overview of CoinPrune's design. We then elaborate on its Bitcoin-retrofittable block-pruning scheme in Section VII, its handling of application-level data in Section VIII, and its integration into Bitcoin in Section IX. We discuss the security of CoinPrune in Section X and evaluate its performance in Section XI. Section XII concludes this paper.

II. Bitcoin Overview

We begin with a primer on Bitcoin before describing its transaction management, its cache of relevant transaction outputs, implications of non-financial transaction data, the deployment of consensus updates, and the bootstrapping process.

Bitcoin Primer. Bitcoin's [1] blockchain provides a public and immutable append-only ledger of financial transactions to prevent the double-spending of coins within an untrusted peer-to-peer (P2P) network. Bitcoin establishes this ledger by bundling pending transactions in cryptographically interlinked, hard-to-create blocks. At its core, the P2P network consists of two special types of nodes. First, *full nodes* maintain the blockchain by locally validating all pending transactions and proposed new blocks and by discarding any incorrect data. Second, *miners* invest their hardware resources to find new blocks while solving a proof-of-work (PoW) puzzle in exchange for freshly minted bitcoins as a reward. Modifying blocks at a later point becomes increasingly hard as it requires recomputing all subsequent blocks to keep their chaining intact. On a technical level, Bitcoin blocks consist of a *header* and a set of transactions. The 80 B-long header contains the block's version, the hash value of the block's predecessor for chaining blocks together, the root of a Merkle tree over the block's transactions to cryptographically tie them to the block header, the time the block was mined, and the miner's PoW.

Transaction Management. Each transaction transfers previously received or newly minted bitcoins to one or more receivers via individual *transaction outputs*. To prevent double-spending, full nodes have to verify any claimed coin ownership for all pending transactions using a dedicated scripting language [25]. Each transaction output defines a spending condition using this script language, which typically sends coins to a specific *Bitcoin address* that corresponds to a cryptographic key pair. A user can prove that she owns the coins associated with a transaction output by providing the data required to satisfy the spending condition in a *transaction input* of a subsequent transaction. For efficiency reasons, all nodes keep track of the current *set of unspent transaction outputs (UTXO set)*. Thereby, nodes can discard pending transactions that attempt to spend non-existing or already spent bitcoins. Notably, by default, full nodes prune all spent transactions, i.e., transactions that do not contribute to the UTXO set anymore, from their transaction index. Nevertheless, these nodes retain a full copy of all blocks for bootstrapping new nodes.

UTXO Set Layout. When processing incoming transactions, each node adds all those transaction outputs to the UTXO set that are not *provably* unspendable. For instance, the standard means of transferring funds in Bitcoin is via pay-to-public-key-hash (P2PKH) transaction outputs, which implement the aforementioned sending of funds to a specific Bitcoin address. The spending condition associated with a P2PKH transaction output requires the designated spender to present the public key matching a cryptographic hash value given in the P2PKH script and a digital signature created using the corresponding private key. However, nodes receiving a transaction have no way to verify that this private key is indeed known to any user [14]. Therefore, nodes are forced to include potentially unspendable transaction outputs in their UTXO set, which would then reside there forever. For a more succinct representation of the UTXO set, nodes *compress* simple and recurring scripts within the UTXOs, e.g., P2PKH scripts that always have the same 25 B-long pattern, with only the 20 B-long hash value changing. Consequently, nodes only store the hash value in their UTXO set and decompress the original transaction output on the fly when validating a

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021).
Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

transaction spending this specific UTXO. In total, Bitcoin Core distinguishes six compressible script types (P2PKH, P2SH, and P2PK with four different types of public keys) and stores any other potentially spendable UTXO in uncompressed form.

Non-Financial Data. Besides the aforementioned financial transactions, users can also permanently store non-financial data on Bitcoin's blockchain. Even though the technical methods for inserting such data vary [12], content is typically inserted in one of the following two ways. First, using a special script starting with the `OP_RETURN` operation in one output allows the user to augment her transaction with up to 80 B of arbitrary data. As such transaction outputs are provably unspendable, nodes do not add them to their UTXO set. Second, other users manipulate the mutable values of standard financial transactions, e.g., hash values within P2PKH transaction outputs, as an unintended means to insert arbitrary data at a higher capacity [12]. As this manipulation cannot be detected reliably and the affected transaction outputs cannot be determined to be unspendable [14], such transaction outputs are added to the UTXO set and are likely to remain there forever. Consequently, any objectionable content may not only enter the immutable blockchain but also reside in the UTXO set indefinitely. We hence refer to this phenomenon as *pollution* of the UTXO set for the remainder of this paper.

Updates of Consensus Rules. Bitcoin must tolerate and resolve blockchain *forks*, i.e., situations in which the blockchain diverges into more than one potential path forward due to its distributed nature. Forks can occur either accidentally, due to the concurrent mining process, or with intent. Accidental forks dissolve naturally since Bitcoin nodes only consider the fastest-growing branch to be valid, and one branch is highly likely to grow faster than other competing branches. Intentional forks are used to *update* existing consensus rules, and they are traditionally categorized as either *hard forks* or *soft forks* [23]. While hard forks introduce protocol-breaking changes, e.g., altered block structures, soft forks aim to remain backward-compatible with clients following older consensus rules [23]. Both paradigms can incur *permanent* blockchain forks depending on whether the majority of nodes accept or reject the proposed changes [23]. Contrarily, multiple works recently investigated *velvet forks* [22], [23], which aim for the *gradual introduction* of new features without creating permanent forks. This type of fork augments upgraded blocks such that they remain valid to legacy nodes, but updated nodes process them according to the changed protocol.

Initial Synchronization. When a node first joins the Bitcoin network, it needs to obtain its individual view on Bitcoin's current state of consensus, i.e., the UTXO set that results from the blockchain path containing the most PoW. To keep this process fully decentralized and independent from trusted nodes, each node initially establishes eight outgoing connections to random other nodes, called *neighbors*, and downloads the complete blockchain from them. Due to the separation of headers and transactions, nodes first fetch the *headerchain*, i.e., the chain of block headers, and simultaneously request full blocks, i.e., the corresponding transactions. While receiving the data, the joining node verifies its correctness by (a) verifying the blockchain's cryptographic links back to the hard-coded genesis block, (b) keeping track of the amount of performed PoW to remain on the currently valid blockchain, (c) validating the transaction sets tied to each block, and (d) verifying and replaying all transactions to obtain an up-to-date UTXO set. Even though the headerchain and the UTXO set are sufficient to process new blocks, the nodes keep a full blockchain copy by default to be able to bootstrap other nodes.

III. Impact of Long-Term Blockchain Utilization

Blockchains continuously grow in size by design and thus eventually reach prohibitive sizes. For instance, Bitcoin's blockchain currently has a size of 281 GiB with an average growth rate of 143 MiB/day as of Sep 29, 2020 [10]. This worrying trend severely impacts the scalability of the overall system. In addition to increasing *storage requirements*, which already exclude whole device classes such as smartphones from operating a full node, there are additional influences on the *required bandwidth*, *processing costs*, and *synchronization times* of joining nodes. Finally, the immutability property may cause pollution with unwanted data over time.

Storage Requirements. To retain a decentralized consensus within its network, Bitcoin requires that enough independent nodes maintain a full blockchain copy to be able to bootstrap joining nodes (cf. Section II). However, storing hundreds of Gigabytes of past blockchain data is irrational for individual node operators and prohibitive for storage-constrained devices. Consequently, such devices cannot act as full nodes, and users have to accept weakened security guarantees.

Bandwidth Requirements. During the initial synchronization, each joining node must obtain a full blockchain copy. Current blockchain sizes already require good Internet connectivity for both the joining node and its neighbors, potentially causing increased initial synchronization times for joining nodes. Furthermore, such requirements also put an additional burden onto existing nodes as serving new nodes consumes resources that could otherwise be used for other tasks, e.g., gossiping pending transactions or newly mined blocks.

Processing Costs. In addition to downloading the blockchain, joining nodes also need to verify the blockchain's integrity and locally replay every single transaction to build the UTXO set. This process consumes excessive amounts of computation power for joining nodes [26]–[28]. In particular, large numbers of obsolete transactions that do not contribute to the UTXO set anymore waste valuable resources.

Synchronization Time. High bandwidth requirements and high processing costs combined cause prolonged synchronization times. While benchmarks report synchronization times of about 5–8 h between 2018 and 2020 [27], [28], literature already highlighted this issue in 2016 when four days were required to synchronize Amazon EC2 nodes [9]. This problem aggravates over time as new blocks are added continuously.

Blockchain Pollution. Blockchain immutability ensures that agreed-upon events or transactions cannot be altered at a later point. However, undesired data, such as manipulated transactions holding illicit content, cannot be removed retrospectively either, and such data also may persistently bloat the UTXO set, leading to UTXO set pollution (cf. Section II). On

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021).
Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

the one hand, a bloated UTXO set implies further scalability issues during the verification of transactions [29]. On the other hand, illicit blockchain content is known to imply legal risks for blockchain users [12].

In summary, the ever-growing blockchain implies strong scalability issues for both joining and established nodes, and the presence of illicit blockchain content presents blockchain nodes with further security challenges. Established nodes are even punished for altruistically serving the full blockchain to help joining nodes synchronize. Especially systems such as Bitcoin, which experience a long-term utilization, suffer from these problems already today. In the following, we thus survey to which extent (newly proposed) systems tackle these issues.

IV. The Current State of Blockchain Pruning

Learning from the observed scalability issues, developers of new blockchain systems tackled these challenges from different perspectives. In this section, we survey current state-of-the-art measures deployed to existing systems and alternative blockchain designs that focus on reducing storage requirements and improving the bootstrapping. Finally, we provide an overview of further related work that does not primarily aim to improve the bootstrapping process.

A. Survey Criteria and Methodology

We qualitatively assess the applicability and effectiveness of related approaches based on (a) their scalability improvements, (b) their capability to maintain sufficient security levels, (c) their impact on the overall network, (d) their potential impact on blockchain queryability, and (e) their compatibility with already established public blockchains, such as Bitcoin.

We assess *scalability improvements* by considering processing, traffic, and storage improvements separately and, from this, we infer the impact on synchronization times for joining nodes. We resort to a qualitative assessment of the presented approaches, as most works do not provide comparable performance benchmarks. Furthermore, we discuss to which extent the improvements impact their base systems' *security* guarantees. The *impact on the overall network* engulfs consequences for the *network health*, i.e., its dependency on especially altruistic nodes and potential additional overheads imposed on established nodes. Then, we assess how the proposed schemes may impact the *blockchain queryability*, e.g., the capability of querying now-obsolete transactions or augmented transactions such as Bitcoin's `OP_RETURN` transactions. Finally, we survey their *compatibility* with already deployed blockchain systems. We summarize our results in Table I.

B. Measures Deployed in Existing Blockchain Systems

The increased popularity of cryptocurrencies forced their developers to tackle rising scalability issues. We now discuss measures taken either locally by users or network-wide by blockchain developers. Our discussion is based on the reference implementations (Bitcoin Core and Ethereum's `geth`, respectively) where appropriate. Overall, we identify approaches based on *trust delegation*, *skipping verification* steps, *improving data management* with the special case of *block pruning*, and *state-based synchronization*.

Trust Delegation. Users rely on third parties to root trust in the blockchain's correctness if they cannot operate a full node, e.g., when using a constrained device for issuing transactions. Using *hot wallets* [30], users essentially outsource all fund management to a trusted third party, enabling the service provider to issue transactions on their behalf. Similarly, *light nodes* [31] outsource blockchain verification to other full nodes, but they manage their wallet locally using simplified payment verification (SPV) [1]. These approaches vastly improve the performance of clients, only put a negligible burden on the full nodes, and they are actively used. However, clients do not contribute positively to the overall network using these approaches as they only seize other nodes' resources. Contrarily, trust-delegating nodes heavily rely on a backbone network of full nodes for both trust and relevant information and prohibit local verifiability. The never-deployed Ultimate Compression scheme [32] aimed at bootstrapping light nodes with the current UTXO set but requires full nodes to store and transmit a searchable representation of the UTXO set in addition to its full blockchain copy, putting an extra burden on the full nodes. Furthermore, this scheme requires an additional blockchain to establish trust in the transmitted UTXO set.

Improving Data Management. Increasing blockchain sizes necessitate optimized data management, either for looking up relevant information or for bootstrapping new nodes efficiently. To this end, Bitcoin Core has historically changed its *underlying database* system [33] and the internal layout of its UTXO set [34]. Furthermore, full nodes *locally prune* obsolete entries from their *transaction index* [33]. While the nodes still persist the full raw blockchain data, such obsolete information is not queryable anymore. Network-related optimizations mainly engulf a revised *header-first download* of the blockchain [35]. Verifying the headerchain is sufficient to ensure the blockchain's integrity. Since transactions can be decoupled from their block's headers (cf. Section II), nodes can now download and verify full blocks in parallel with only minor and local upgrading incompatibilities [35]. They can further *limit their block-serving bandwidth* [46] and relay *compact representations* of newly mined blocks, which avoids broadcasting known-but-pending transactions redundantly [47]. However, the header-first download still requires nodes to obtain and process all blockchain data during their initial synchronization, and only improves the distribution of that data.

Skipping Verification. Early on, Bitcoin's reference implementation started to avoid revalidating transactions from very old blocks. Using hard-coded *checkpoint blocks* at first [36], Bitcoin has recently shifted to use configurable *assumed-valid blocks* [37]. The reasoning here is that invalid transactions would have been rejected by the network earlier, and thus older transactions with many confirmations are believed to be correct. By skipping assumed-valid blocks, joining nodes can avoid the costly signature verification of large portions of the blockchain at negligible security risks. However, joining nodes still download the complete blockchain to replay all transactions to create an up-to-date UTXO set.

Simple Block Pruning. To counter increasing storage requirements, Bitcoin users have the option to completely *prune*

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021).
Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

TABLE I
QUALITATIVE COMPARISON OF APPROACHES IMPROVING STORAGE REQUIREMENTS AND INITIAL SYNCHRONIZATION

	Name	Approach	Reduce Processing	Reduce Traffic	Reduce Storage	Sync. Time	Maintain Security	Network Health	Server Burden	Completeness	Compatibility
Deployed Solutions	Hot Wallets [30]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	○	○	●	○	●
	Light Nodes [31]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	◐	○	●	○	●
	"Ultimate Compression" [32]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	○	●	○	●	●
	DB Improvements [33], [34]	Data Management	◐	○	○	◐	●	●	●	●	●
	Index Pruning [33]	Data Management	◐	○	◐	◐	●	●	●	◐	●
	Headers-first Download [35]	Data Management	○	○	○	◐	●	●	●	●	◐
	Assume-valid Blocks [36], [37]	Skip Verification	●	○	○	◐	●	●	●	●	●
	Block Pruning [15]	Simple Block Pruning	○	○	●	○	●	○	●	◐	●
	Ethereum Fast Sync [38]	State-based Sync.	●	○	○	●	●	●	●	●	○
	AssumeUTXO [39]	State-based Sync.	○	○	○	●	◐	●	●	●	●
Related Work	Selective Pruning [40]	Simple Block pruning	◐	◐	●	◐	○	●	○	●	○
	Rollerchain [19]	State-based Sync.	●	●	●	●	●	●	◐	◐	○
	Marsalek et al. [41]	State-based Sync.	○ / ●	○ / ●	○ / ●	●	●	◐	◐	◐	○
	Mini Blockchain Scheme [17]	Balance-based Sync.	●	●	●	●	●	◐	◐	○	○
	Mimblewimble [18]	Balance-based Sync.	○	◐	◐	○	●	●	●	○	○
	Pascal [20]	Balance-based Sync.	●	●	●	●	◐	○	●	◐	○
	Vault [42]	Balance-based Sync.	●	●	●	●	●	●	●	◐	○
	FlyClient [43]	Commitment-based	○ / ●	○ / ●	○ / ●	○ / ●	●	○	●	○	○
	TICK [44]	Commitment-based	○ / ●	○ / ●	○ / ●	○ / ●	●	○	●	○	○
	MiniChain [45]	Commitment-based	●	●	●	●	●	●	●	○	○
	CoinPrune (our approach)	State-based Sync.	●	●	●	●	●*	●*	◐	◐	●

□ / □: *Distinction Full Nodes / Light Nodes* ∗: *Dependent on honest majority among adopters (cf. Section X)*

raw blockchain data [15] after a *full* initial synchronization. This step allows nodes to forget all obsolete blockchain data at the cost of its queryability. In contrast to local index pruning, block pruning is detrimental to the network health as block-pruning nodes are incapable of bootstrapping new nodes.

State-based Synchronization. While Bitcoin focuses on financial transactions, other cryptocurrencies, such as Ethereum [8], can also execute smart contracts. Naturally, those cryptocurrencies have more complex state layouts as the full nodes need to keep track of every smart contract's state. Consequently, Ethereum uses *Fast Sync* [38], which enables joining nodes to download a recent state and avoid replaying all blockchain data. However, Ethereum still values the queryability of older data. Thus, joining nodes also download and persist all blocks but they do not have to process them during their initial synchronization. In contrast to Bitcoin's proposal for Ultimate Compression, Fast Sync remains secure since Ethereum, by default, cryptographically ties its current state to each block [8]. Hence, nodes can verify the correctness of their obtained state directly via Ethereum's blockchain. Since other cryptocurrencies lack these header fields, Fast Sync is not immediately portable. AssumeUTXO [39] aims for a similar extension for Bitcoin by extending upon its assume-valid blocks. Joining nodes use state-based synchronization based on a recent UTXO set to be operable early on, but then transition to a full background synchronization to retain full queryability. Furthermore, AssumeUTXO relies on hard-coded cryptographic state identifiers as well as third-party servers for the state distribution, which affects its security.

Takeaway. Developers have tackled the scalability issues of blockchain systems from different perspectives. However, all approaches either have limited efficiency or questionable security properties, they are detrimental to the network health, or they are not readily portable to already established systems.

C. Proposed Block-Pruning Schemes

The lacking efficiency of post-deployment measures motivated various *alternative blockchain designs* that promise improved scalability measures. We identify alternative designs that refine mere *block-pruning schemes*, designs proposing *state-based* or *balance-based* synchronization, and *commitment-based* designs aiming to further reduce or even remove the need for locally maintaining the current state.

Simple Block Pruning. Palm et al. [40] present a distributed block-pruning scheme for established nodes in permissioned blockchains, i.e., blockchains jointly maintained by a fixed set of mutually known parties. A dedicated pruning initiator defines a pruning algorithm that all nodes must execute to identify and prune now-irrelevant transactions in a way that other nodes can still retrieve all relevant data. However, this approach focuses on permissioned blockchains and requires a dedicated initiator. Hence, the approach is inapplicable to public settings, which are open to unknown or unauthenticated parties, both for security and compatibility reasons.

State-based Synchronization. Similar to Ethereum's Fast Sync, and inheriting its advantages and disadvantages, Rollerchain [19] proposes a state-based bootstrapping process. However, Rollerchain values performance over complete queryability, thereby significantly decreasing synchronization overheads as old information can be fully pruned. Similarly, Marsalek et al. [41] propose a state-based bootstrapping process based on Bitcoin, but the approach forfeits compatibility with Bitcoin by rejecting blocks with invalid states attached.

Balance-based Synchronization. A special class of state-based block-pruning schemes simplifies the structure of what constitutes the state to allow for more efficient representations and updates [17], [18], [20], [42]. Typically, these schemes only keep track of existing accounts and their balances. The Mini-Blockchain scheme [17] replaces Bitcoin's UTXO set with an account tree that is cryptographically tied to each mined block. Joining nodes obtain the headerchain and a recent account tree to synchronize before fully processing a tail of full blocks to preserve PoW-based security. However, the scheme expects established nodes to compute slices of the recent account tree on demand without elaborating how it ensures the availability of all required data to rewind the account tree accordingly. Mimblewimble [18] follows a similar approach but emphasizes confidential transactions at the cost of synchronization performance as joining nodes

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021). Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

have to obtain and verify rangeproofs for unspent funds [18]. Through their balance-based approach, both schemes limit the expressiveness of transactions. To overcome this limitation, Pascal [20] defines SafeBoxes as a replacement for mere account trees. SafeBoxes permit the generation of a limited number of accounts per block and are designed to enable higher-layer applications. However, the limited availability of account spots is conceptually detrimental to network health. Finally, Vault [42] builds upon Algorand [48] to enable the distribution of fragments of recent states across the network to reduce the per-node storage requirements. Therefore, Vault is inapplicable as an aid for existing, simpler cryptocurrencies.

Commitment-based Improvements. Recently, multiple approaches proposed to extend the commitment to the current state within block headers and make state data queryable this way [43]–[45]. FlyClient [43] reduces a client's overhead when relying on SPV by avoiding the need to obtain the full headerchain from other nodes to convince the client that its contact node relies on a valid blockchain. FlyClient achieves this goal by maintaining a Merkle mountain range [49] over the headerchain. A Merkle mountain range is an append-only variant of a Merkle tree that allows for efficient membership tests [43]. In FlyClient, a light node uses this commitment in each block to probabilistically challenge full nodes about the correctness of their headerchain before accepting their blocks and engage in SPV [43]. TICK [44] instead encodes the UTXO set as an AVL hash tree that commits to the UTXO set in a way that (a) changes to the UTXO set, i.e., insertion and deletion of UTXOs when processing a new block, can efficiently be reflected based on an existing AVL hash tree and (b) light nodes can efficiently verify the presence or absence of a specific UTXO based on an up-to-date AVL hash tree [44]. Both approaches assume the presence of full nodes that store the full UTXO set but that can provide light nodes with a succinct proof to convince them of their honesty. Contrarily, MiniChain [45] aims for a practical *stateless blockchain* that overcomes the need for storing the full UTXO set for verification purposes. Based on prior work by Boneh et al. [50], MiniChain uses RSA-based accumulators suited for the distributed setting to commit to the UTXO set in a similar way to TICK. However, MiniChain requires fund spenders to provide an additional proof to convince validating nodes that the transaction only spends existing coins that have not previously been spent [45]. Similarly to balanced-based pruning schemes, these commitments are, however, incompatible with already deployed cryptocurrencies.

Takeaway. Alternative blockchain designs have shown that incorporating cryptographic ties to recent state objects are a promising means to establish trust in state-based blockchain synchronization processes. However, extending existing systems with such capabilities immediately results in hard forks, which are difficult to deploy and thus highly debated [21].

D. Further Related Work

Other works that consider blockchain data management include analyses of blockchain data [11]–[13], [29], [51]–[53] and the UTXO set [54], lightweight payment schemes [55], [56], approaches to prevent illicit content from being engraved into the blockchain [14], [16], [57]–[60], and sharding approaches [61]–[63]. In the following, we provide pointers to cover the research perspectives for this further related work.

Previous Analyses. Bitcoin's blockchain and scalability properties have previously been analyzed from different perspectives. Initial work considered the transaction graph, e.g., in the quantitative analysis due to Ron and Shamir [51] or the assessment of payment anonymity by Meiklejohn et al. [52]. Several works surveyed scalability limitations and corresponding remedies [64]–[67]. Higher-level data semantics were the subject of further analyses. After Shirriff [68] highlighted the presence of non-financial blockchain content hidden within transactions, Matzutt et al. [11], [12] further formalized this aspect with a quantitative content analysis with a focus on potentially objectionable content on Bitcoin's blockchain. Sward et al. [13] concurrently investigated more sophisticated content insertion methods. Meanwhile, Bartoletti et al. [29], [53] had investigated the constructive utilization of `OP_RETURN` data. Delgado-Segura et al. [54] and Pérez-Solà et al. [69] have investigated the UTXO sets of Bitcoin and related cryptocurrencies. Finally, Lu et al. [70] applied game theory to investigate the potential for even lighter clients.

Lightweight Payment Schemes. To scale to large transaction throughputs, one branch of related work aims to avoid on-chain transactions to the largest extent possible without losing the trustworthiness of payments. Poon and Dryja presented the Bitcoin Lightning Network [55] for this purpose, which since has sparked several further works [56], [71]–[74].

Countering Illicit Content. Matzutt et al. [14] have assessed how to hinder the insertion of blockchain content in Bitcoin with only minimal changes to the consensus rules. Most related countermeasures deal with illicit content after it has been inserted into the blockchain. Florian et al. [16] propose that nodes locally remove content they deem objectionable and investigate how to outsource the validation of removed transactions to other nodes. A different branch of research revolves around redactable blockchains, which allow to remove transactions either based on the decision of one or few redactors [57], removal votings among miners [59], or based on redaction policies specified by the transaction owner herself [60], [75], e.g., to deal with privacy concerns.

Blockchain Sharding. One special measure to increase the scalability of blockchain systems is *sharding*. In traditional blockchain systems, all nodes redundantly replay all events locally. Hence, they do not have to especially trust other nodes at the cost of limited scalability. Blockchain sharding, contrarily, partitions the control over the blockchain over time. Bitcoin-NG [76] initially proposed to repurpose the mining process such that a successful miner becomes a leader capable of subsequently issuing blocks at a much higher frequency until the mining process yields the next leader. Following this proposal, Luu et al. [61] proposed to use entropy periodically derived from blockchain data to randomly elect small committees that are subsequently responsible for processing a subset of transactions on behalf of the whole network. Further work, such as OmniLedger [63] and RapidChain [62], has improved upon this initial design since.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021).
Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

Takeaway. Lightweight payment schemes provide an orthogonal approach to tackle blockchain scalability by avoiding on-chain transactions. Further research also investigates illicit content on blockchains. Corresponding countermeasures, except for the local erasure proposed by Florian et al. [16], usually neglect the possibility of objectionable content also entering the UTXO set, i.e., the state maintained even when pruning a blockchain. Finally, the epochs typically used in sharding blockchains have proven themselves a valuable building block for synchronizing multiple nodes and can be reused, e.g., for coordinating periodically recurring tasks.

V. Requirements for a Secure and Retrofittable Block-Pruning Scheme

Blockchain systems require that sufficiently many nodes maintain a *full local blockchain copy* (cf. Section II). While this initial design becomes massively burdening for these nodes as well as joining nodes, multiple approaches to fully pruning obsolete data have been proposed (cf. Section IV-C). However, none of these approaches can be adapted to directly provide similar optimizations for established systems (e.g.,, Bitcoin) without provoking major incompatibilities. Furthermore, to the best of our knowledge no current pruning scheme considers *data semantics*, e.g., UTXO set pollution or higher-level application data. To realize *fully compatible* extensions for the network-wide pruning of obsolete data in existing cryptocurrencies while *maintaining already established security levels* and enabling *additional protection*, we identify the following requirements and design goals:

(G1) Scalability. To be effective, pruning schemes must provide improvements for *all* metrics discussed in Section III, i.e., storage and bandwidth demands for joining and block-serving nodes, processing costs, and synchronization time.

(G2) Correctness. Starting from the genesis block, each joining node must obtain the same local state, with or without the block-pruning scheme enabled, to ensure that the network's consensus about accepted transactions is kept intact. In particular, the node must learn about all accepted, non-obsolete events, and it must not accept any false events.

(G3) Verifiability. As security is a top priority, pruning schemes must keep joining nodes capable of verifying the correctness of the synchronization process even in the presence of adversaries. Here, we require that block-pruning schemes do not reduce the security of the overall blockchain system.

(G4) Compatibility. Popular and long-living blockchain systems are especially affected by scalability limitations. Instead of proposing new systems (cf. Section IV-C), all changes should be applicable to existing blockchains, especially Bitcoin, even during operation. Preferably, the scheme is opt-in, e.g., as achievable via velvet forks (cf. Section II).

(G5) Data Semantics. Block-pruning schemes may assume more tasks and responsibilities than simply pruning obsolete data. One crucial aspect is the handling of non-financial blockchain data. Block-pruning schemes should be aware of the semantics of such data, i.e., mitigate negative impacts of UTXO set pollution where possible and avoid impeding higher-level applications that make use of (prunable) `OP_RETURN` transaction outputs.

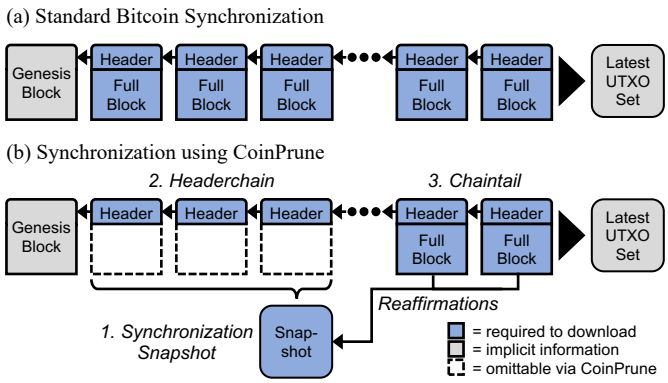


Fig. 1. High-level design overview of CoinPrune. Instead of downloading and verifying all full blocks, joining nodes obtain a recent snapshot in a trustworthy manner due to its on-chain reaffirmations by multiple miners.

VI. CoinPrune Overview

To address the challenges resulting from the ever-increasing size of existing blockchains, we present *CoinPrune*, our secure, snapshot-based block-pruning scheme that is gradually deployable without protocol-breaking changes, e.g., to Bitcoin. In this section, we overview CoinPrune's design and, hereafter, we present its block-pruning scheme and the way we handle additional data semantics in the following sections.

CoinPrune is designed to transfer scalability improvements of novel blockchain designs (cf. Section IV-C) to Bitcoin while keeping compatibility a priority. We now describe how *CoinPrune nodes*, i.e., Bitcoin nodes that additionally support CoinPrune, jointly maintain recent snapshots on the blockchain and how new nodes can bootstrap securely via those snapshots. Finally, we outline benefits for the whole network.

Snapshot Maintenance. CoinPrune nodes periodically create *snapshots* of their UTXO set. These snapshots are served to joining nodes instead of the full blockchain history for reduced storage, bandwidth, and processing requirements **(G1)**. They are tied to the current *block height*, i.e., the position of the most recent block in the blockchain, and contain a well-ordered UTXO set for synchronization **(G2)** and verification **(G3)** purposes, respectively. To prevent malicious nodes from distributing invalid snapshots, e.g., in an attempt to multiply their funds, CoinPrune requires snapshots to be *publicly announced* to the blockchain by referencing a cryptographic *identifier* of each snapshot on-chain. CoinPrune miners place these announcements in their blocks' *coinbase transactions*, which the miners issue to mint new coins. By utilizing an existing field within the coinbase transactions, which may contain 100 B of arbitrary data, we keep CoinPrune Bitcoin-compatible **(G4)**. Other CoinPrune miners independently do the same, which causes nodes deriving snapshots from the same UTXO set to *mutually reaffirm* that snapshot's validity. This approach creates positive-only feedback, i.e., wrong snapshots are not rejected but tolerated and outpaced by valid snapshots' reaffirmations given an honest majority of CoinPrune miners.

Bootstrapping Nodes. Instead of downloading all blockchain data, a joining node can now securely bootstrap in three steps, as shown in Figure 1: First, the node obtains a *recent snapshot* either from its neighbors or through a snapshot-offering third party (cf. Section X). For P2P-based

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021). Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

snapshot acquisition, the node downloads the snapshot advocated by the majority of its neighbors. Second, the node obtains the *headerchain*, i.e., the interconnected and lightweight block headers, to learn about the blockchain branch with the most PoW in it. Third, the node downloads the *chaintail*, i.e., the full blocks following the snapshot's block height. Via the chaintail, the joining node can (a) catch up with recent transactions and (b) inspect the full blocks for snapshot reaffirmations. If the joining node observes sufficiently many reaffirmations of the obtained snapshot, it accepts this snapshot and concludes the initial synchronization process. Otherwise, the node discards the insecure snapshot and retries while reconnecting to new neighbors.

Global Block Pruning. Since joining nodes can securely bootstrap from the headerchain, the snapshot, and the chaintail, *all* CoinPrune nodes may now *safely prune older blocks* from before the snapshot. As new snapshots are reaffirmed periodically, nodes may also prune aging snapshots as well without hurting the network health. Single *archival nodes* may still keep a full blockchain copy to retain the full and reliable queryability of also older data.

Additional Data Semantics. CoinPrune is aware that (a) entries in the UTXO set may have been manipulated to cause UTXOs to be unspendable or even to hold illicit content and that (b) small application-level data chunks stored in `OP_RETURN` transactions are desirable to preserve but inherently prunable. To account for these higher-level data semantics **(G5)**, CoinPrune nodes can *locally obfuscate* most manipulable identifiers without impeding the nodes' capability of validating pending transactions and they maintain an additional *application data storage*, which contains all `OP_RETURN` transaction outputs and their context (cf. Section VIII).

VII. Retrofittable Block Pruning

We now present CoinPrune's block-pruning scheme, which is designed for a retrospective deployment to existing cryptocurrencies. We discuss the data management of CoinPrune nodes, how these nodes coordinate the pruning process, and how new nodes can now bootstrap efficiently.

A. Adapted Data Management

To understand CoinPrune in detail, we discuss the layout of its snapshots and the required changes to the nodes' local data management stemming from the pruning of older blocks.

Snapshot Creation. Each snapshot corresponds to a specific block height, meaning that it represents a serialization of the UTXO set obtained from processing all blocks up to and including that height. A CoinPrune snapshot consists of a simple header and multiple chunks of serialized UTXOs, and it is referenced on-chain by a cryptographic identifier. The header holds the snapshot's corresponding block height, that block's identifier, and the number of chunks in the snapshot. The identifier is a special hash value created over the snapshot's header and chunks to uniquely represent the snapshot in a succinct manner. First, the header and each chunk are hashed individually using Bitcoin's HASH256 function (SHA256 applied twice). Then, the snapshot identifier is the hash value of the concatenation of these hash values. Using this simple

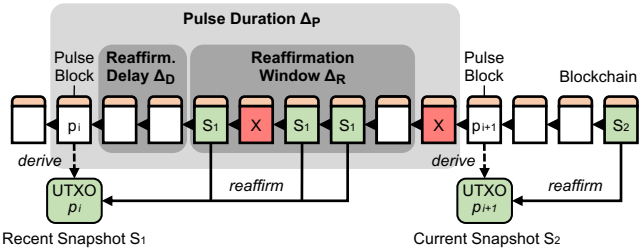


Fig. 2. Our pulse-based coordination triggers the creation of new snapshots. Any invalid or delayed reaffirmations are ignored.

snapshot serialization, joining nodes are immediately aware of all available chunks and can independently request individual chunks from different neighbors in parallel. Further, we limit chunk sizes to 1 MB akin to Bitcoin's maximum block size.

Persisted Information. By shifting to a snapshot-based synchronization process, nodes may now prune older data by deleting the full blocks prior to the snapshot's block height. However, the nodes must remain capable of serving the full headerchain to joining nodes. Before pruning blocks, these nodes thus need to persist some information currently held by Bitcoin's block index. These are block identifiers, headers, block heights, the amount of PoW, the number of transactions, and the block's timestamp. Persisting this data, a recent snapshot, and the chaintail of not-yet-prunable full blocks is now sufficient to securely bootstrap joining nodes.

B. Coordination of CoinPrune Nodes

CoinPrune establishes trust in recent snapshots by having miners mutually reaffirm these snapshots' correctness. Joining nodes then use these reaffirmations as their trust anchor when deciding whether to synchronize based on a particular snapshot. As shown in Figure 2, CoinPrune miners coordinate based on *pulse blocks* p_i, which are issued in constant intervals Δ_P, e.g., every 10 000 blocks. These pulse blocks trigger the creation of a new snapshot and its subsequent mutual reaffirmation at fixed positions on the blockchain, i.e., all nodes act based on the same information.

Each snapshot is attached to a pulse block, i.e., CoinPrune nodes subsequently reaffirm the snapshot derived from the pulse block's corresponding UTXO set. All CoinPrune miners reaffirm the last pulse block's attached snapshot by adding the snapshot's identifier to their blocks' coinbase fields during a relatively short *reaffirmation window* Δ_R. During Δ_R, multiple concurring snapshot identifiers might occur if an adversary attempts to get an invalid snapshot reaffirmed. However, we assume an honest majority among CoinPrune miners, just like in the overall Bitcoin network, and thus the genuine snapshot is expected to accumulate reaffirmations the fastest. We further enforce an *acceptance threshold* k, i.e., a joining node only accepts the most reaffirmed snapshot if it was reaffirmed at least k times during Δ_R. If no snapshot reaches k reaffirmations during Δ_R, this pulse is invalid, and pruning is delayed until the next pulse starts. The goals behind these measures are (a) preventing a dishonest minority from outpacing an honest majority during snapshot reaffirmation and (b) preventing a dishonest majority stemming from low overall CoinPrune support from successfully reaffirming any

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021). Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

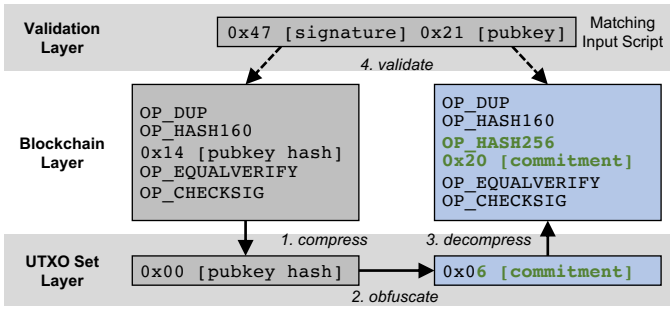


Fig. 3. CoinPrune obfuscates hash-based, compressed UTXOs (here P2PKH) by additionally applying `OP_HASH256` to the script's mutable value. Nodes account for this change during decompression and can still validate the expected input script spending the obfuscated UTXO without further alterations.

snapshot during Δ_R (cf. Section X). Finally, the reaffirmation window may be delayed by a small *reaffirmation delay* Δ_D, e.g., $\Delta_D = 6$ blocks, to ensure that accidental forks affecting the pulse block p_i are resolved before creating a snapshot.

C. Bootstrapping New Nodes

The reaffirmations periodically published on Bitcoin's blockchain allow joining nodes to bootstrap as follows. First, the node obtains a recent snapshot. The node can now securely acquire a recent snapshot from its neighbors using off-chain P2P requests or through external means, e.g., mirror servers. Second, the joining node downloads and verifies the header-chain from its neighbors to learn about the blockchain branch with the most PoW in it, as is already done by Bitcoin [35]. Third, instead of downloading and processing all blockchain data, the joining node applies the previously obtained snapshot in good faith to initially fill its UTXO set. Finally, the joining node fetches and processes the *chaintail*, i.e., the remaining full blocks succeeding the snapshot's block height, to finalize synchronizing its UTXO set. During this full synchronization phase, the joining node additionally inspects the chaintail's coinbase transactions for reaffirmations of its applied snapshot. If the node learns that its snapshot was the most-reaffirmed one during Δ_R and that it was reaffirmed at least k times, it accepts the snapshot and concludes the bootstrapping step. Otherwise, the joining node aborts and obtains a different snapshot from another source, e.g., by connecting to a new set of neighbors.

VIII. Handling Application-Level Data

We now present CoinPrune extensions for handling non-financial data. We first show how local *obfuscation* effectively removes illicit contents from the UTXO set without losing Bitcoin compatibility. Then, we discuss how CoinPrune preserves intended application-level data, e.g., `OP_RETURN` data.

A. Obfuscation of Illicit Data from the UTXO Set

As shown by Florian et al. [16], nodes can locally delete objectionable content from their copy of the blockchain and their UTXO set at the cost of outsourcing the validation of corresponding transactions to other nodes. We extend upon this idea and integrate a simple *UTXO obfuscation scheme* into CoinPrune that prevents nodes from recovering any objectionable data from snapshots while remaining capable of using obfuscated UTXOs during transaction validation.

Construction. In previous work, we proposed *self-verifying blockchain identifiers* [14] to harden the mutable fields of a transaction output script against easy content insertion. Essentially, this measure forces transaction creators to only release *one-way commitments* to these mutable values while allowing full nodes to detect further tampering easily. As indicated by Florian et al. [16], CoinPrune can make use of similar and simple local UTXO set obfuscation, which extends to the snapshots sent to joining nodes: Hash-based transaction outputs, such as P2PKH (cf. Section II), already commit to a later disclosed public key by recording only its HASH160 value on-chain, i.e., the spender needs to open the commitment as part of the spending condition. However, because this on-chain commitment remains mutable and has been used for content insertion [12], we *locally commit* to these values again *before* including the results in the UTXO set.

More specifically, CoinPrune locally obfuscates mutable and potentially content-holding blockchain identifiers, as shown in Figure 3. Before updating the UTXO set, each Bitcoin node already compresses any transaction output corresponding to one of the six predefined patterns (cf. Section II) and uses one byte to denote the compression case applied, i.e., the values `0x00` to `0x05` (Step 1). Otherwise, the node keeps the uncompressed script and prepends it with its length plus six (to account for the six compression cases). Using our obfuscation scheme (Step 2), CoinPrune-supporting nodes now additionally apply Bitcoin's HASH256 function to the respective mutable values in any standard-compliant, hash-based transaction output. Consequently, the node cannot restore the value due to the pre-image resistance of SHA256. Besides P2PKH, CoinPrune can thus also obfuscate pay-to-script-hash (P2SH) scripts, which commit to an entire script defining a more flexible spending condition, and their segregated witness-based counterparts P2WPKH and P2WSH [77] this way. Combined, these script types account for 99.2 % of all current UTXOs (cf. Section XI-B). We account for the proper decompression of obfuscated values to maintain compatibility with Bitcoin by introducing four new compression cases, e.g., an obfuscated P2PKH value will be denoted by `0x06` instead of `0x00` (cf. Figure 3). During decompression (Step 3), the node can now detect obfuscated values and add the required `OP_HASH256` operation as the second local commitment layer. This way, the node remains capable of validating incoming input scripts even if the spender is unaware of the obfuscation.

Limitations. Our obfuscation scheme relies on the fact that most UTXOs already contain a commitment instead of, e.g., raw public keys, i.e., the transaction input spending a UTXO will open the commitment. In this case, we can indeed add another commitment layer without further changes. However, albeit deprecated, Bitcoin still permits P2PK and P2MS transaction outputs, which contain one or multiple raw public keys, respectively [78]. Since the main goal of CoinPrune is remaining retrofittable to Bitcoin, we cannot adapt the required transaction input scripts. Furthermore, we refrain from caching raw public keys before committing to them in the UTXO set since these public keys can also be manipulated [12]. Hence, our scheme cannot support the obfuscation of these or any non-standard transaction outputs without losing its deployability

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021). Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

as a velvet fork. Still, our obfuscation scheme already covers 99.2 % of the current UTXOs (cf. Section XI-B).

Future Potentials. In case of widespread adoption of CoinPrune, it may become possible to shift away from ensuring deployability as a velvet fork and propose further changes affecting the UTXO set. First, strictly enforcing transaction standardness and rejecting legacy P2PK and P2MS transaction outputs can improve the coverage of our obfuscation scheme. Second, currently non-obfuscatable UTXOs already contained in the UTXO set could be rewritten to become obfuscatable. This rewriting is currently not possible since legacy nodes would not be aware that they have to create a different script when attempting to spend their old coins. Finally, if the network agrees on carefully rewriting the UTXO set to a certain extent, future work can investigate further means to counteract UTXO set pollution. For instance, nodes could completely drop UTXOs flagged for encoding illicit content or repossess old entries that only hold negligible values and thereby bloat the UTXO set. However, such modifications would have strong implications for Bitcoin in general, demanding a comprehensible yet minimal rule set for allowed modifications approved by a large majority of nodes.

B. Preservation of Application-Level Data

Bitcoin explicitly enables the insertion of small, application-specific data chunks in a transaction using `OP_RETURN` transaction outputs (cf. Section II). Such transaction outputs are inherently unspendable, and thus they are not included in the UTXO set. Consequently, block-pruning schemes, such as CoinPrune, would remove this data and thereby break higher-level applications, e.g.,, audit systems [2]–[4].

Thus, a second CoinPrune extension provides an *application data storage* that realizes a lookup table of past `OP_RETURN` transaction outputs with their context (block and transaction). Each entry in this lookup table consists of (a) the `OP_RETURN` payload as well as the identifiers of the corresponding (b) transaction and (c) block. As nodes still maintain the headerchain, this information suffices to associate application data with a specific transaction and its inclusion time.

Applying this extension changes the semantics of a CoinPrune reaffirmation. The application data storage is maintained in 1 MB-chunks just as snapshots are (cf. Section VII-A), and thus nodes obtain its identifier in the same way. However, the reaffirmation tags written to the blockchain are now derived by applying HASH256 to the concatenated snapshot identifier *and* the application data storage's identifier.

IX. Seamless Integration into Bitcoin

CoinPrune's main feature is its immediate applicability to Bitcoin **(G4)**. In this section, we present our means to achieve *gradual opt-in deployability* to Bitcoin via a velvet fork, assuming that a sufficient share of honest miners makes a rational choice to support CoinPrune, e.g., to preserve storage.

On-Chain Data. Although snapshot reaffirmations must be publicly announced on Bitcoin's blockchain, CoinPrune's utilization of only a block's coinbase field prevents any protocol-breaking changes. Full nodes that are unaware of CoinPrune will ignore any snapshot reaffirmation, and CoinPrune nodes will never reject blocks containing incorrect reaffirmations. Instead, they will try to outpace incorrect reaffirmations with legitimate ones. Hence, our scheme fulfills the requirements for a gradually deployable velvet fork [22], [23]. To prevent CoinPrune nodes from confusing snapshot reaffirmations with other coinbase data, we propose to encapsulate the reaffirmation accordingly, e.g., using a unique prefix and separators such as `CoinPrune/[reaffirmation]/`. While CoinPrune has to potentially share the coinbase field with other applications, e.g., merged mining [79], its messages remain distinguishable from messages of other applications.

Peer-to-Peer Protocol. Even though CoinPrune allows for external snapshot sources, most joining nodes will likely rely on Bitcoin's network to obtain their initial snapshot. To enable this Bitcoin-intrinsic snapshot acquisition, we extend Bitcoin's P2P protocol [80] with an additional `GETSTATE` message type sent by CoinPrune nodes. Joining nodes send a `GETSTATE` message to each neighbor to learn about available recent snapshots. Each neighbor responds with an inventory (`INV` message) that contains the hash values of the snapshot header and the chunks of their most recent available and successfully reaffirmed snapshot as `STATE` objects. The joining node uses these `INV` messages to determine which snapshot to obtain and derives that snapshot's identifier. Then, the node requests individual chunks of the most-advertised snapshot from its neighbors using sequences of `GETDATA` messages. Finally, the node applies the snapshot once all chunks are available. For increased compatibility, we restrict chunk sizes to 1 MB, i.e., Bitcoin's old maximum block size, and we introduce a new service flag for Bitcoin's `VERSION` handshake to avoid sending unknown messages to CoinPrune-unaware nodes.

X. Security Discussion

In this section, we discuss how CoinPrune helps nodes bootstrap correctly and securely **(G2)** based on verifiable snapshots **(G3)**, which are maintained by an honest majority of CoinPrune miners. We first show that the on-chain reaffirmations created by our scheme reliably reference snapshots from arbitrary sources to be used for bootstrapping joining nodes more efficiently. Second, we discuss the influence of the positive-only feedback created by CoinPrune miners mutually reaffirming the current snapshot has on its trustworthiness. We extend our previous work [24] by an empirical simulation, showing that CoinPrune can achieve similar security properties as Bitcoin if at least 32 % of Bitcoin nodes support the scheme. Finally, we discuss the P2P-level security of CoinPrune.

Verifying Snapshot Validity. CoinPrune's on-chain reaffirmations enable joining nodes to reliably verify a snapshot's correctness. To this end, CoinPrune uses a cryptographic hash function in a layered fashion to obtain snapshot identifiers, thereby hashing each snapshot chunk individually. This approach enables joining nodes to notice any manipulation of individual chunks, e.g., performed by malicious nodes, during their initial synchronization. As the snapshot identifier also covers the snapshot's metadata and the hash values of all chunks, joining nodes can further derive at which block height the snapshot was created and verify that they obtained all original chunks. Thus, adversaries cannot deceive joining

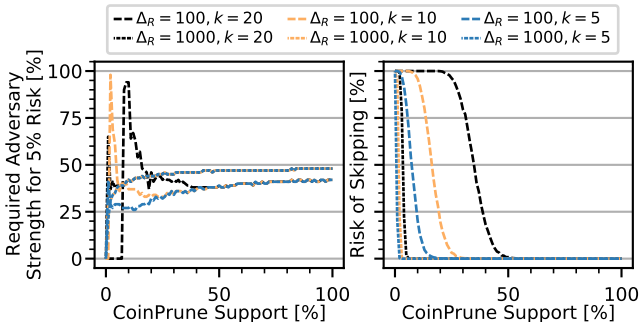


Fig. 4. Larger reaffirmation windows increase CoinPrune's resilience against adversaries independent from the acceptance threshold.

nodes to accept invalid or inconsistent snapshots, e.g., to reinstate spent UTXOs. Consequently, nodes can safely apply valid snapshots regardless of their source, i.e., the Bitcoin network or a third party, such as a snapshot-mirroring website.

Adversary Resilience. The reaffirmation tags CoinPrune miners add to their blocks allow joining nodes to verify that an obtained snapshot has indeed been reaffirmed. However, an adversary may attempt to reaffirm an invalid snapshot. In Section VII-B, we introduced the acceptance threshold k and the reaffirmation window Δ_R to prevent an adversary from successfully reaffirming an invalid snapshot. To investigate the effectiveness of these means, we simulated a random mining process of 1000 active miners with an increasing support of CoinPrune among miners and an increasing adversarial influence among CoinPrune miners.

Namely, we consider a support of CoinPrune among miners of $0\,\% \leq f_C \leq 100\,\%$ that grows in increments of $1\,\%$. For each sampling point, we assume a growing fraction f_A of these CoinPrune miners to be controlled by an adversary, where we again vary $0\,\% \leq f_A \leq 100\,\%$ in increments of $1\,\%$. For instance, if $f_C = 25\,\%$ and $f_A = 10\,\%$, then we assume that 250 out of 1000 active miners reaffirm a snapshot, but 25 of those miners will reaffirm an invalid one. We repeated each simulation 1000 times and considered reaffirmation windows $\Delta_R \in \{100, 1000\}$ and acceptance thresholds $k \in \{5, 10, 20\}$, respectively, and investigated how often (a) a joining node would accept the correct snapshot, (b) the adversary was able to successfully reaffirm an invalid snapshot, and (c) no snapshot was reaffirmed successfully within the reaffirmation window, i.e., the pulse is skipped without pruning.

In Figure 4, we show the minimum fraction of adversaries f_A required such that at least $5\,\%$ of snapshots a joining node accepts are in fact reaffirmed by the adversary (left-hand side) and the worst-case risk over all f_A that no snapshot was reaffirmed successfully (right-hand side). Our results show that a longer reaffirmation window of $\Delta_R = 1000$, i.e., roughly one week, raises the required f_A for compromising at least $5\,\%$ of reaffirmed snapshots to $f_A \geq 46\,\%$ for $f_C \geq 31\,\%$ with $f_A = 48\,\%$ for full CoinPrune support independent of the acceptance threshold k. For the smaller Δ_R, increasing k has the desired effect of raising the required f_A in case of low CoinPrune adoption, but this comes at the cost of disproportionally impeding the operation of CoinPrune due to a large number of skipped pulses. We can counteract this effect by decreasing the pulse duration Δ_P, and thus a *dynamic* approach adapting Δ_R and Δ_P based on a previously sampled support level f_C is promising to combine the positive effects of short and long reaffirmation windows. During phases of low support, e.g., $f_C < 10\,\%$, we can operate with $\Delta_R = 100$, but aggressively retry to reaffirm snapshots by also setting $\Delta_P = \Delta_R = 100$. If f_C was sufficiently large during the last pulse, we can relax CoinPrune's aggressiveness and set, e.g., $\Delta_R = 1000$ and $\Delta_P = 10\,000$. Finally, a voting phase among miners, similar to that preceding the rollout of P2SH support [81], can ensure a sufficient initial adoption rate f_C.

P2P Attacks. Nodes announce their support of CoinPrune during the version handshake (cf. Section IX) to integrate well with Bitcoin's P2P protocol **(G4)**. Our scheme further adds only two message types not understood by vanilla Bitcoin nodes, which are only exchanged with CoinPrune nodes. First, joining nodes send an additional `GETSTATE` message. Second, contacted nodes answer with an `INV` message containing the newly introduced `STATE` objects. Both message types are in line with the design of Bitcoin's P2P protocol, and considerations regarding its DoS resilience or Eclipse attacks [82] translate directly to CoinPrune. Furthermore, as shown above, an adversary cannot undetectably advertise an invalid or partially manipulated snapshot. Contrarily, the P2P layer provides an additional defense layer in the rare case where an adversary controlling only a minority of CoinPrune miners successfully reaffirms an invalid snapshot by chance. In this case, honest nodes will not advertise the invalid snapshot in their `INV` message and thus provoke a pulse to be effectively skipped instead of spreading invalid snapshots. Whenever a joining node suspects an attack, it will start over and bootstrap with a new neighbor set. When reconnecting one of our commodity PCs (cf. Section XI-A) to the Bitcoin network every 30 min between Jan 26 and Feb 1, 2021 it took the client ~68 s on average to connect to eight new neighbors. While the client continually establishes new connections afterward, it establishes its tenth outgoing connection after only ~154 s on average, which shows the feasibility of our approach even when accounting for early disconnects.

Takeaway. Through CoinPrune, joining nodes can securely obtain the snapshots for initial synchronization from arbitrary sources given their on-chain reaffirmations' reliability.

XI. Performance Evaluation

We now demonstrate that CoinPrune enables massive performance savings for Bitcoin nodes **(G1)**. After describing our testbed setup, we present the storage savings achieved for all nodes. Further, we show that traffic and synchronization times for joining nodes are massively reduced as well.

A. Testbed Setup for Synchronization Measurements

We use our proof-of-concept implementation of CoinPrune based on Bitcoin Core v0.17.1 to run measurements on a server (2× Intel Xeon E5-2630 v4, 32 GB RAM, 8 TB Seagate IronWolf ST8000VN0022-2EL112), which bootstraps from eight identical commodity PCs (Intel Core2 Quad Q9400, 8 GB RAM, 500 GB Hitachi Deskstar 7K500) running CoinPrune as well. All devices are interconnected via a Linksys SLM2024 Gigabit switch. For our measurements, we consider snapshots

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021). Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

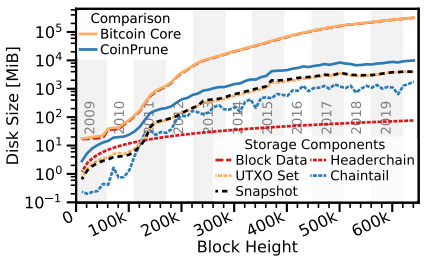


Fig. 5. Currently, our scheme already reduces storage requirements by two orders of magnitude.

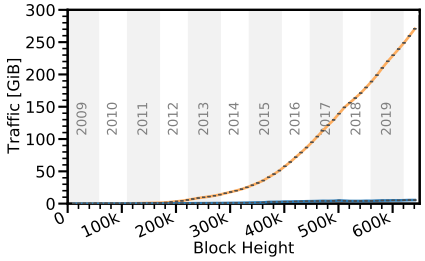


Fig. 6. CoinPrune allows bootstrapping with vastly reduced amounts of traffic, unburdening all nodes.

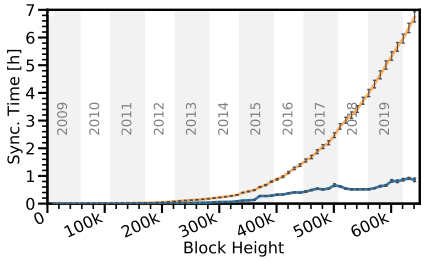


Fig. 7. Our scheme reduces initial synchronization times, yet the snapshot size impacts performance.

being created every 10 000 blocks up until a block height of 640 000 (Jun 17, 2020) with 1000 additional blocks (~1 week of blocks) as our chaintail. We perform the synchronization using vanilla Bitcoin Core in one go and synchronize via CoinPrune based on the aforementioned snapshots. While storage requirements are fully determined by the blockchain data, synchronization times and traffic may vary. Hence, the latter were averaged over five independent runs, and error bars denote the 99 % confidence intervals. We omitted to check the coinbase field for the snapshot identifier to be able to use Bitcoin's real blockchain for our measurements.

B. Storage Savings

CoinPrune allows all Bitcoin nodes to prune older blocks in exchange for maintaining a recent snapshot and the headerchain to serve joining nodes. In Figure 5, we depict how the main contributors to Bitcoin's storage demand changed over time compared to the serialized snapshot and headerchain required to operate CoinPrune. From this, we derive the overall storage requirements for Bitcoin Core and CoinPrune, respectively. For Bitcoin's storage requirements, we consider the heavily dominating `blocks` folder containing raw block data, information required to rewind blocks efficiently, and the block index, as well as the `chainstate` folder holding the UTXO set. In contrast to this, CoinPrune needs to store one serialized snapshot and the serialized headerchain, as well as the UTXO set and chaintail for live operation. Our measurements show that the sizes of serialized snapshots align well with those of Bitcoin's UTXO set. Minor variances stem from different encodings of both data structures. Further, persisting the headerchain to reconstruct Bitcoin's block index comes at only negligible costs of 133.33 B per block, resulting in a headerchain size of 76.29 MiB for our latest measurement. Finally, considering block heights starting from 500 000, the chaintail has an average size of 1.09 GiB. Overall, CoinPrune nodes could thus historically reduce their storage requirements by 86.98 %, with the largest absolute and relative savings, currently 302.53 GiB, at higher block heights. These savings account for a decrease of two orders of magnitude, with the potential for becoming even larger as the blockchain grows.

Overhead of Content Obfuscation. We consider our most recent snapshot (block height 640 000), which contains roughly 66.2 m UTXOs, and assess our obfuscation scheme's impact to protect against unwanted content **(G5)**. Using our scheme, we can obfuscate 99.2 % of all UTXOs, the vast majority of which hold P2PKH or P2PSH scripts. However, the obfuscation of these scripts increases the snapshot size since we now store 32 B-long commitments instead of the 20 B-long values. The considered snapshot grows from 3.90 GiB to 4.63 GiB, i.e., we inflict an overhead of 18.72 %. However, the overall storage savings and the protection against unwanted or even illegal blockchain content outweigh this overhead.

Application Data Storage. To assess the growth of the application data storage **(G5)**, we serialized all `OP_RETURN` transaction outputs up to the block height of our most recent snapshot according to our scheme presented in Section VIII-B. This serialized application data storage has a total size of 9.20 GiB and contains roughly 45.2 m entries with an average size of 231 B each. Hence, by pruning old blocks while preserving application data specifically, CoinPrune continues to support higher-level applications at feasible costs.

C. Evaluation of Synchronization Performance

Pruning obsolete data not only relieves Bitcoin nodes from storage depletion but also joining nodes benefit from a widespread adoption of CoinPrune. As shown in Figure 6, the reduced storage requirements directly translate to a reduction in traffic required to synchronize with the Bitcoin network. For instance, synchronizing from a snapshot on block height 640 000 with a chaintail length of 1000, joining nodes only inflict 5.51 GiB of traffic (dominated by the snapshot and the chaintail) when using CoinPrune, whereas legacy nodes would cause 270.70 GiB of traffic to bootstrap successfully. Over the whole blockchain, achievable savings average at 92.99 %. Joining nodes currently have to obtain two orders of magnitude less data during initial synchronization using CoinPrune, which is largely dominated by acquiring the snapshot.

A similar trend can be observed for the overall synchronization time of joining nodes, i.e., obtaining and verifying the headerchain, the snapshot, and the chaintail. Figure 7 shows that CoinPrune improves synchronization times over Bitcoin's whole history, resulting in savings of 77.03 % on average for joining nodes. Even though Bitcoin mitigates revalidating very old transactions due to its assumed-valid blocks (cf. Section IV-B), joining nodes still must replay the whole transaction graph. Contrarily, CoinPrune avoids this step as well due to its reliance on snapshots. Consequently, CoinPrune currently enables joining nodes to catch up with the Bitcoin network in 51 min instead of 7 h using standard Bitcoin. This time increases when a node joins toward the end of a pulse due to a longer chaintail. However, Figure 7 indicates that joining nodes benefit from a vastly improved initial synchronization performance even in this case. This time saving is especially beneficial as initial synchronization is often considered a major scalability concern [9], [28].

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021). Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.

Takeaway. The snapshot-based approach of CoinPrune unburdens both established and joining nodes from major overheads stemming from Bitcoin's bootstrapping process, including the required storage, traffic, and synchronization times. Hence, CoinPrune establishes a secure and effective means to vastly improve Bitcoin's long-term durability.

XII. Conclusion

CoinPrune provides a gradually deployable and secure block-pruning scheme for established cryptocurrencies such as Bitcoin. CoinPrune-supporting miners periodically create snapshots of Bitcoin's current UTXO set and mutually reaffirm the correct snapshot by cryptographically linking it to the blocks they mine. Our scheme hinders an adversary's effort to distribute invalid snapshots by restricting reaffirmation activities to a short window after the snapshot creation and by requiring a minimum number of reaffirmations for joining nodes to accept a snapshot. Our simulation shows that our scheme achieves security properties comparable to Bitcoin when $1/3$ of nodes in the Bitcoin network support CoinPrune. Consequently, all CoinPrune-supporting nodes can completely prune obsolete data and reduce their required storage from currently 312 GiB to 10 GiB (a decrease of 302 GiB), with potentially even larger saving potentials as the blockchain grows. Due to the vastly reduced data to obtain and process, joining nodes can currently reduce their synchronization time by 77 % in our measurements. Finally, CoinPrune handles non-financial data by (a) preserving application-level data from `OP_RETURN` transaction outputs and (b) obfuscating almost all objectionable and potentially illegal data that may still be present in the UTXO set after pruning.

ACKNOWLEDGEMENTS. This work has been funded by the German Federal Ministry of Education and Research (BMBF) under funding reference numbers 16KIS0443, 16DHLQ013, and Z31 BMBF Digital Campus. The funding under reference number Z31 BMBF Digital Campus has been provided by the German Academic Exchange Service (DAAD). The responsibility for the content of this publication lies with the authors.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," White paper, 2008.
[2] Proof of Existence, "Proof of Existence," 2015, accessed 2021-02-02. https://proofofexistence.com
[3] V. Durham, "Namecoin," 2011, accessed 2021-02-02. https://namecoin.org
[4] M. Henze, B. Wolters *et al.*, "Distributed Configuration, Authorization and Management in the Cloud-based Internet of Things," in *IEEE TrustCom/BigDataSE/ICESS*, 2017.
[5] M. Chase and S. Meiklejohn, "Transparency overlays and applications," in *ACM CCS*, 2016.
[6] K. Nikitin, E. Kokoris-Kogias *et al.*, "CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds," in *USENIX Security*, 2017.
[7] R. Matzutt, J. Pennekamp *et al.*, "Utilizing Public Blockchains for the Sybil-Resistant Bootstrapping of Distributed Anonymity Services," in *ACM ASIACCS*, 2020.
[8] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," White paper, 2016.
[9] K. Croman, C. Decker *et al.*, "On Scaling Decentralized Blockchains," in *IFCA FC Bitcoin Workshop*, 2016.
[10] Blockchain.com, "Blockchain Charts," 2011, accessed 2021-02-02. https://www.blockchain.com/charts
[11] R. Matzutt, O. Hohlfeld *et al.*, "POSTER: I Don't Want That Content! On the Risks of Exploiting Bitcoin's Blockchain as a Content Store," in *ACM CCS*, 2016.
[12] R. Matzutt, J. Hiller *et al.*, "A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin," in *IFCA FC*, 2018.
[13] A. Sward, I. Vecna, and F. Stonedahl, "Data Insertion in Bitcoin's Blockchain," *Ledger*, vol. 3, 2018.
[14] R. Matzutt, M. Henze *et al.*, "Thwarting Unwanted Blockchain Content Insertion," in *IEEE IC2E Workshops*, 2018.
[15] Bitcoin Project, "Bitcoin Core version 0.11.0 released," 2015, accessed 2021-02-02. https://bitcoin.org/en/release/v0.11.0
[16] M. Florian, S. Henningsen *et al.*, "Erasing Data from Blockchain Nodes," in *IEEE EuroS&PW*, 2019.
[17] J. D. Bruce, "The Mini-Blockchain Scheme," White paper, 2014.
[18] A. Poelstra, "Mimblewimble," White paper, 2016.
[19] A. Chepurnoy, M. Larangeira, and A. Ojiganov, "Rollerchain, a Blockchain With Safely Pruneable Full Blocks," White paper, 2016.
[20] H. Schoenfeld and A. Molina, "Pascal: An Infinitely Scalable Cryptocurrency," White paper, 2019.
[21] D. Morgan, "The Great Bitcoin Scaling Debate – A Timeline," 2017, accessed 2021-02-02. https://hackernoon.com/the-great-bitcoin-scaling-debate-a-timeline-6108081dbada
[22] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," *IACR Cryptology ePrint Archive*, vol. 2017/963, 2017.
[23] A. Zamyatin, N. Stifter *et al.*, "A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice," in *IFCA FC Bitcoin Workshop*, 2018.
[24] R. Matzutt, B. Kalde *et al.*, "How to Securely Prune Bitcoin's Blockchain," in *Proc. IFIP Networking*, 2020.
[25] Bitcoin Project, "Script," 2010, accessed 2021-02-02. https://en.bitcoin.it/wiki/Script
[26] Bitcoin Project, "Running a Full Node," 2015, accessed 2021-02-02. https://bitcoin.org/en/full-node
[27] M. Geniar, "Initial impressions on running a Bitcoin Core full node," 2019, accessed 2021-02-02. https://ma.ttias.be/initial-impressions-on-running-a-bitcoin-core-full-node
[28] J. Lopp, "2020 Bitcoin Node Performance Tests," 2020, accessed 2021-02-02. https://blog.lopp.net/2020-bitcoin-node-performance-tests
[29] M. Bartoletti, B. Bellomy, and L. Pompianu, "A Journey into Bitcoin Metadata," *J Grid Computing*, vol. 1, pp. 3–22, 2019.
[30] Bitcoin Project, "Hot Wallet," 2012, accessed 2021-02-02. https://en.bitcoin.it/wiki/Hot_wallet
[31] Bitcoin Project, "Lightweight Node," 2018, accessed 2021-02-02. https://en.bitcoin.it/wiki/Lightweight_node
[32] A. Reiner, "Ultimate Blockchain Compression w/ Trust-free Lite Nodes," 2012, accessed 2021-02-02. https://bitcointalk.org/index.php?topic=88208
[33] Bitcoin Project, "Bitcoin-Qt version 0.8.0 released," 2013, accessed 2021-02-02. https://bitcoin.org/en/release/v0.8.0
[34] Bitcoin Project, "Bitcoin Core version 0.15.0 released," 2017, accessed 2021-02-02. https://bitcoin.org/en/release/v0.15.0
[35] Bitcoin Project, "Bitcoin Core version 0.10.0 released," 2015, accessed 2021-02-02. https://bitcoin.org/en/release/v0.10.0
[36] S. Nakamoto, "Bitcoin 0.3.2 released," 2010, accessed 2021-02-02. https://bitcointalk.org/index.php?topic=437
[37] Bitcoin Project, "Bitcoin Core version 0.14.0 released," 2017, accessed 2021-02-02. https://bitcoin.org/en/release/v0.14.0
[38] P. Szilágyi, "eth/63 fast synchronization algorithm," 2015, accessed 2021-02-02. https://github.com/ethereum/go-ethereum/pull/1889
[39] J. O'Beirne, "Assume UTXO," 2019, accessed 2021-02-02. https://github.com/jamesob/assumeutxo-docs/tree/master/proposal
[40] E. Palm, O. Schelén, and U. Bodin, "Selective Blockchain Transaction Pruning and State Derivability," in *IEEE CVCBT*, 2018.
[41] A. Marsalek, T. Zefferer *et al.*, "Tackling Data Inefficiency: Compressing the Bitcoin Blockchain," in *IEEE TrustCom/BigDataSE*, 2019.
[42] D. Leung, A. Suhl *et al.*, "Vault: Fast Bootstrapping for the Algorand Cryptocurrency," in *ISOC NDSS*, 2019.
[43] B. Bünz, L. Kiffer *et al.*, "FlyClient: Super-Light Clients for Cryptocurrencies," in *IEEE EuroS&P*, 2020.
[44] W. Zhang, J. Yu *et al.*, "TICK: Tiny Client for Blockchains," *IEEE Internet of Things Journal*, 2020, early Access version.
[45] H. Chen and Y. Wang, "MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 67–76, 2020.
[46] Bitcoin Project, "Bitcoin Core version 0.12.0 released," 2016, accessed 2021-02-02. https://bitcoin.org/en/release/v0.12.0
[47] Bitcoin Project, "Bitcoin Core version 0.13.0 released," 2016, accessed 2021-02-02. https://bitcoin.org/en/release/v0.13.0
[48] Y. Gilad, R. Hemo *et al.*, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *ACM SOSP*, 2017.

[49] P. Todd, "Merkle Mountain Ranges," accessed 2021-02-02. https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md

[50] D. Boneh, B. Bünz, and B. Fisch, "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains," in *Springer CRYPTO*, 2019.

[51] D. Ron and A. Shamir, "Quantitative Analysis of the Full Bitcoin Transaction Graph," in *IFCA FC*, 2013.

[52] S. Meiklejohn, M. Pomarole *et al.*, "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names," in *ACM IMC*, 2013.

[53] M. Bartoletti and L. Pompianu, "An analysis of Bitcoin OP_RETURN metadata," in *IFCA FC Bitcoin Workshop*, 2017.

[54] S. Delgado-Segura, C. Pérez-Solà *et al.*, "Analysis of the Bitcoin UTXO set," in *IFCA FC Bitcoin Workshop*, 2018.

[55] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," White paper, 2016.

[56] M. Green and I. Miers, "Bolt: Anonymous Payment Channels for Decentralized Currencies," in *ACM CCS*, 2017.

[57] G. Ateniese, B. Magri *et al.*, "Redactable Blockchain – or – Rewriting History in Bitcoin and Friends," in *IEEE EuroS&P*, 2017.

[58] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to forget without hard forks," *IACR Cryptology ePrint Archive*, vol. 2017/106, 2017.

[59] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable Blockchain in the Permissionless Setting," in *IEEE S&P*, 2019.

[60] A. Dorri, S. S. Kanhere, and R. Jurdak, "MOF-BC: A Memory Optimized and Flexible Blockchain for Large Scale Networks," *Future Gener. Comput. Syst.*, vol. 92, 2019.

[61] L. Luu, V. Narayanan *et al.*, "A Secure Sharding Protocol For Open Blockchains," in *ACM CCS*, 2016.

[62] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *ACM CCS*, 2018.

[63] E. Kokoris-Kogias, P. Jovanovic *et al.*, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *IEEE S&P*, 2018.

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

[65] A. A. Monrat, O. Schelén, and K. Andersson, "A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities," *IEEE Access*, vol. 7, 2019.

[66] Q. Zhou, H. Huang *et al.*, "Solutions to Scalability of Blockchain: A Survey," *IEEE Access*, vol. 8, 2020.

[67] A. Hafid, A. S. Hafid, and M. Samih, "Scaling Blockchains: A Comprehensive Survey," *IEEE Access*, vol. 8, 2020.

[68] K. Shirriff, "Hidden surprises in the Bitcoin blockchain and how they are stored," 2014, accessed 2021-02-02. http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html

[69] C. Pérez-Solà, S. Delgado-Segura *et al.*, "Another coin bites the dust: An analysis of dust in UTXO-based cryptocurrencies," *Royal Society Open Science*, vol. 6, no. 1, 2019.

[70] Y. Lu, Q. Tang, and G. Wang, "Generic Superlight Client for Permissionless Blockchains," Cryptology ePrint Archive, Report 2020/844, 2020, https://eprint.iacr.org/2020/844.

[71] C. Decker and R. Wattenhofer, "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels," in *Springer SSS*, 2015.

[72] R. Khalil and A. Gervais, "Revive: Rebalancing Off-Blockchain Payment Networks," in *ACM CCS*, 2017.

[73] G. Malavolta, P. Moreno-Sanchez *et al.*, "Concurrency and Privacy with Payment-Channel Networks," in *ACM CCS*, 2017.

[74] A. Miller, I. Bentov *et al.*, "Sprites and State Channels: Payment Networks that Go Faster Than Lightning," in *IFCA FC*, 2019.

[75] D. Derler, K. Samelin *et al.*, "Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based," in *NDSS*, 2019.

[76] I. Eyal, A. E. Gencer *et al.*, "Bitcoin-NG: A Scalable Blockchain Protocol," in *USENIX NSDI*, 2016.

[77] E. Lombrozo, J. Lau, and P. Wuille, "BIP 141: Segregated Witness (Consensus Layer)," 2016, accessed 2021-02-02. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki

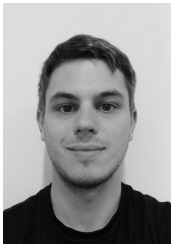
[78] K. Okupski, "Bitcoin Developer Reference," White paper, 2014.

[79] A. Judmayer, A. Zamyatin *et al.*, "Merged Mining: Curse or Cure?" in *DPM, CBT*, 2017.

[80] Bitcoin Project, "P2P Network," 2017, accessed 2021-02-02. https://developer.bitcoin.org/devguide/p2p_network.html

[81] G. Andresen, "BIP 16: Pay to Script Hash," 2012, accessed 2021-02-02. https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki

[82] E. Heilman, A. Kendler *et al.*, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *USENIX Security*, 2015.



Roman Matzutt received the B.Sc. and M.Sc. degrees in Computer Science from RWTH Aachen University. He is a researcher at the Chair of Communication and Distributed Systems (COMSYS) at RWTH Aachen University. His research focuses on the challenges and opportunities of accountable and distributed data ledgers, especially those based on blockchain technology, and means allowing users to express their individual privacy demands against Internet services.

Benedikt Kalde received the B.Sc. degree in Computer Science from Paderborn University and the M.Sc. degree in Computer Science from RWTH Aachen University. In his master thesis with the Chair of Communication and Distributed Systems (COMSYS) he helped conceptualize and analyze CoinPrune's initial design and implemented its first prototype. Today, he evaluates use cases of emerging technology for a consultancy in the aviation industry.

Jan Pennekamp received the B.Sc. and M.Sc. degrees in Computer Science from RWTH Aachen University with honors. He is a researcher at the Chair of Communication and Distributed Systems (COMSYS) at RWTH Aachen University. His research focuses on security & privacy aspects in the Industrial Internet of Things (IIoT). In particular, his special interests include privacy-enhancing technologies, the design of privacy-preserving protocols, and secure computations as well as their application.

Arthur Drichel received the B.Sc. and M.Sc. degrees in Computer Science from RWTH Aachen University. He is a researcher at the Research Group IT-Security at RWTH Aachen University. His research interests include intrusion detection systems, privacy enhancing technologies, and machine learning.

Martin Henze received the Diploma (equiv. M.Sc.) and PhD degrees in Computer Science from RWTH Aachen University. He is a postdoctoral researcher at the Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE, Germany. His research interests lie primarily in the area of security and privacy in large-scale communication systems, recently especially focusing on security challenges in the industrial and energy sectors.

Klaus Wehrle received the Diploma (equiv. M.Sc.) and PhD degrees from University of Karlsruhe (now KIT), both with honors. He is full professor of Computer Science and Head of the Chair of Communication and Distributed Systems (COMSYS) at RWTH Aachen University. His research interests include (but are not limited to) engineering of networking protocols, (formal) methods for protocol engineering and network analysis, reliable communication software, and all operating system issues of networking.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021). Changes were made to this version by the publisher prior to publication. The final version of record is available at http://doi.org/10.1109/TNSM.2021.3073270.