

Roman Matzutt

**Demystifying and Adjusting the Promises
of Blockchain-based Data Management
in the Permissionless Setting**

Demystifying and Adjusting the Promises of Blockchain-based Data Management in the Permissionless Setting

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Master of Science

Roman Matzutt

aus Heinsberg

Berichter:

Prof. Dr.-Ing. Klaus Wehrle
Prof. Dr. rer.nat. Frank Kargl

Tag der mündlichen Prüfung: 20.10.2023

Reports on Communications and Distributed Systems

edited by
Prof. Dr.-Ing. Klaus Wehrle
Communication and Distributed Systems,
RWTH Aachen University

Volume 24

Roman Matzutt

Demystifying and Adjusting the Promises of Blockchain-based Data Management in the Permissionless Setting

Shaker Verlag
Düren 2024

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Zugl.: D 82 (Diss. RWTH Aachen University, 2023)

Copyright Shaker Verlag 2024

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 978-3-8440-9516-6

ISSN 2191-0863

Shaker Verlag GmbH • Am Langen Graben 15a • 52353 Düren

Phone: 0049/2421/99011-0 • Telefax: 0049/2421/99011-9

Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

The digital currency Bitcoin introduced the blockchain as a data structure that allows its users to establish consensus about who owns which coins in a decentralized manner. Since then, blockchain technology has evolved and now enables distrusting parties to engage in online interactions without the need for a trusted intermediary by immutably recording general events in transactions. This interaction model sparked a tremendous interest in blockchain technology, its potential, and applications.

However, the identification of multiple shortcomings has since dampened this initial spirit of optimism. These shortcomings are especially apparent for permissionless blockchains, such as Bitcoin, which openly encourage participation by anybody. For instance, Bitcoin has to secure its blockchain against malicious actors by relying on energy-intensive computations, which further leads to scalability issues as only few payments can be accepted at a time. While prior work has extensively studied such technical challenges, it neglected the influence of the data stored on the blockchain so far. Yet, this influence becomes undeniable: On the one hand, unknown actors can irrevocably append new data without a designated removal process. On the other hand, the operation of a blockchain system depends on a massive replication of its full and growing history. Hence, the impact of blockchain-recorded data requires thorough investigation to ensure the security and longevity of blockchain systems.

In this dissertation, we thus take a data-driven perspective to assess and improve the applicability of permissionless blockchains as building blocks for decentralized data management systems. We identify two core challenges of blockchain-based data management, i.e., the need for moderating what data is recorded and the need for alleviating the continually growing storage requirements stemming from the append-only nature of blockchains. Furthermore, we assess the potential of blockchains to enable additional applications by seizing their characteristic properties. We address these challenges on a technical level via the following contributions.

As our first contribution, we systematically analyze the phenomenon of blockchain content insertion on a conceptual, technical, and empirical level. Our analysis reveals that content insertion is a common practice and offers benefits for higher-level applications, but inserting illicit content can potentially create devastating consequences for the participants. As our second contribution, we explore means to mitigate these consequences, both before and after the fact, by proposing strategies to prevent the insertion of unwanted content as well as a redactable blockchain that enables a swift and transparent removal of content. Our third contribution addresses the challenge of growing blockchain sizes by defining a gradually deployable block-pruning scheme that is retrofittable to Bitcoin and enables users to retroactively forget obsolete data and thereby reduce their storage requirements. Finally, our fourth contribution shows that permissionless blockchains still hold untapped potential for fueling novel applications despite their limitations; namely, we demonstrate how Bitcoin can help securely bootstrapping decentralized anonymity services.

Overall, we shed new light on the potential impact of the data persisted on blockchains. Our analyses and technical contributions therefore widen the scope for resilient and durable blockchain designs for data management tasks.

Kurzfassung

Die digitale Wahrung Bitcoin hat die Blockchain als eine Datenstruktur etabliert, mit der Nutzer dezentralisiert Konsens daruber erlangen konnen, wem welche Munzen gehoren. Seitdem hat sich die Blockchaintechnologie stetig weiterentwickelt, so dass nun sich misstrauende Parteien ohne Intermediar uber das Internet interagieren konnen, indem beliebige Ereignisse unwiderruflich aufgezeichnet werden. Dies hat ein enormes Interesse an der Technologie, ihr Potenzial und ihre Anwendungen entfacht.

Allerdings hat die Entdeckung einiger Nachteile diese Aufbruchstimmung derweil gedampft. Diese Nachteile kommen insbesondere bei den frei zuganglichen Permissionless Blockchains, wie Bitcoin, zum Tragen. Beispielsweise muss Bitcoin die Blockchain uber energieintensive Berechnungen gegen Angreifer absichern, was zu Skalierbarkeitsproblemen fuhrt, da so nur wenige Zahlungen auf einmal akzeptiert werden konnen. Obwohl solche technischen Herausforderungen bereits intensiv studiert wurden, wurde der Einfluss der Daten in der Blockchain bisher vernachlassigt. Dabei ist dieser Einfluss unbestreitbar: Einerseits konnen Unbekannte Daten unwiderruflich und ohne vorgesehenen Loschprozess anhangen. Andererseits bedingt der Betrieb einer Blockchain eine massive Replizierung ihrer gesamten und stetig wachsenden Historie. Daher bedurfen die Auswirkungen der Blockchain-Daten einer sorgfaltigen Analyse, um die Sicherheit und Langlebigkeit dieser Systeme sicherzustellen.

In dieser Dissertation fokussieren wir uns auf ebendiese Daten, um die Eignung von Permissionless Blockchains als Bausteine fur dezentralisierte Datenmanagementsysteme zu beurteilen und zu verbessern. Wir identifizieren zwei Kernherausforderungen des Blockchain-basierten Datenmanagements, namlich die Moderierbarkeit der Daten und den Bedarf, die wachsenden Speicheranforderungen aufgrund stetig angehangter Daten abzumildern. Zudem bemessen wir das Potenzial der Blockchain, mittels ihrer spezifischen Eigenschaften weitere Anwendungen zu ermoglichen. Diese Herausforderungen adressieren wir auf technischer Ebene mittels folgender Beitrage.

Als ersten Beitrag analysieren wir das Phanomen des Einfugens von Blockchaininhalten auf konzeptioneller, technischer und empirischer Ebene. Unsere Analyse zeigt, dass dieses Vorgehen gebrauchlich ist und Vorteile fur Anwendungen bietet, jedoch hat das Speichern illegaler Inhalte potenziell verheerende Konsequenzen. Als zweiten Beitrag untersuchen wir Mittel, diese Konsequenzen sowohl im Vorfeld als auch im Nachgang einzudammen, indem wir Strategien, die das Einfugen unerwunschter Inhalte verhindern, und eine editierbare Blockchain, die eine rasche und transparente Loschung erlaubt, vorschlagen. Unser dritter Beitrag adressiert das Problem wachsender Blockchaingroen, indem ein graduell ausrollbares Block-Pruning-Verfahren definiert wird, das Bitcoin-Nutzern nachtraglich ermoglicht, obsoletere Daten zu vergessen und so ihren Speicherbedarf zu reduzieren. Zuletzt zeigt unser vierter Beitrag, dass Permissionless Blockchains noch unerschlossenes Potenzial haben, trotz ihrer Limitationen neue Anwendungen zu realisieren, indem wir demonstrieren, wie Bitcoin das sichere Bootstrapping dezentraler Anonymisierungsdienste unterstutzen kann.

Insgesamt werfen wir ein neues Licht auf die moglichen Auswirkungen der auf Blockchains gespeicherten Daten. Unsere Analysen und technischen Beitrage erweitern so den Raum fur resiliente und bestandige datenzentrierte Blockchainedesigns.

Acknowledgments

In a way, working on a PhD could retrospectively largely be described as sitting in the office most days, being occupied with activities, sometimes exciting, sometimes mundane, and occasionally going to a conference to celebrate the prior acceptance of a paper of hard work. What I mean to imply by this: On paper, this type of work does not sound like the blueprint for a thrilling or moving story. Yet, despite having felt incapable of believing otherwise, I have to come to the conclusion that my personal journey was all but uneventful. I started this journey on the brink of losing a fight against demons I was painfully aware of, yet could not fully grasp them. Now, I am able to write these words and feel genuinely happy.

A major part of this turn is owed to the great persons I met along the way. I will only highlight some of them and definitely forget someone in the following. But I count on you to know your contributions and how much I value them. Thank you.

First off, I want to thank my advisor Klaus Wehrle for giving me the chance to work on my PhD at COMSYS and believing in my capabilities when I could not. The freedom of research you provide has had a significant impact on my personal and professional development and helped me overcome challenges I would not have felt suited for otherwise. Further, I want to thank Frank Kargl for taking on the role of the second opponent during my PhD defense.

My time at COMSYS could have never been as great without my inspirational and great colleagues. This holds especially, but not exclusively, for my partners in crime of the Security & Privacy group. Martin, Henrik, Jens, and René; you helped me kick off and develop my researcher career back when the group was comparably small. I am grateful to have learned from all of you on a professional and personal level. Markus, Ina, Konrad, Eric, Christian, and Johannes; you make up the not-so-small next generation, and I am confident that you will excel at your endeavors. No, I did not forget you, Jan. I will never forget our paper-writing marathon in 2019, it was a great demonstration of how well our different approaches complement each other. And, “besides,” you became a great friend along the way and I could not have wished for anyone else to share an office with. Keep up the great work! Further, I have to thank those who endured my rants and kept up the pragmatism, which sometimes also works wonders in research, who would have known. I am looking at you, Torsten, Jan, Ike, Constantin, and also Claudia! Of course, also everyone whom I did not mention explicitly here contributes their fair share to keeping COMSYS a great place. Besides all other colleagues, this also holds for all the students I had the honor of helping take their first steps, of whom there are too many to name here.

Finally, I want to thank my family for all their support and believing in me. To my mother Lydia, my father Karli, and my brother Nico: Thank you for everything. The same holds for all my supportive friends, you are the best! The best gift of all this time, however, was having the indescribable luck of randomly meeting my wife. Roxanna, I do not know how my life would have turned out without you. But, frankly, I do not care in the slightest. Getting to know you was the single best and most impactful event of my life. Not only am I genuinely happy, but impatiently anticipating what our future will hold for us. I love you.

Contents

1	Introduction	1
1.1	Problem Analysis	3
1.1.1	Promises of Blockchain Technology	3
1.1.2	Problems of Blockchain-based Data Management	5
1.2	Research Questions	8
1.3	Contributions	9
1.3.1	Attribution of Contributions	14
1.4	Outline	16
2	Bitcoin and Blockchain Technology	17
2.1	Bitcoin Overview	18
2.2	Blockchain Structure	23
2.2.1	Overview	23
2.2.2	Block Structure	24
2.2.3	Merkle Trees	27
2.3	Transactions	29
2.3.1	Overview	29
2.3.2	Coin Ownership	31
2.3.3	Scripting System	35
2.3.4	Segregated Witnesses	42
2.3.5	Managing Unspent Transaction Outputs	43
2.4	Consensus	44
2.4.1	Mining Process	45
2.4.2	Distribution and Validation Process	47

2.4.3	Forks	49
2.5	Peer-to-Peer Network	52
2.5.1	Network Overview	52
2.5.2	Node Connections	55
2.5.3	Data Dissemination	56
2.5.4	Initial Synchronization Process	58
2.6	Summary	59
3	Systematic Analysis of Non-Financial Blockchain Content	61
3.1	Motivation	61
3.1.1	Contributions	62
3.2	Problem Analysis	64
3.2.1	History of Bitcoin Content Insertion	64
3.2.2	Model for Blockchain Content Insertion and Distribution	66
3.3	Analysis of Content Insertion Methods	67
3.3.1	Low-Level Content Insertion Methods	68
3.3.2	Content Insertion Services	72
3.4	Benefits and Risks of Blockchain Content	74
3.4.1	Benefits of Blockchain Content Insertion	74
3.4.2	Negative Consequences of Blockchain Content Insertion	75
3.5	Analysis of Non-Financial Content	81
3.5.1	Measurement Framework and Methodology	81
3.5.2	Quantitative Analysis of Content Insertion	86
3.5.3	Assessment of Blockchain Files	95
3.6	Related Work	99
3.7	Conclusion and Future Work	101
4	Mitigation of Unwanted Blockchain Content	103
4.1	Motivation	103
4.1.1	Constraints for Mitigation Schemes	105
4.1.2	Contributions	107
4.2	Prevention Strategies Against Unwanted Content	107

4.2.1	Related Work	109
4.2.2	Problem Statement	110
4.2.3	Filtering Content-Holding Transactions	112
4.2.4	Mandatory Minimum Transaction Fees	114
4.2.5	Hardened On-Chain Addresses	120
4.2.6	Summary and Future Work	124
4.3	Moderation of Blockchain Content	125
4.3.1	Related Work	126
4.3.2	Missing Swift and Transparent Redactions	129
4.3.3	Cryptographic Building Blocks	131
4.3.4	RedactChain Overview	135
4.3.5	Detecting Unwanted Blockchain Content	137
4.3.6	Decentralized Redaction Process	139
4.3.7	Coordinating Short-Lived Redaction Juries	142
4.3.8	Evaluation	145
4.3.9	Summary and Future Work	150
4.4	Conclusion	152
5	Retrofitting Blockchains with Pruning Capabilities	153
5.1	Motivation	154
5.1.1	Problem Analysis	155
5.1.2	Contributions	158
5.2	Survey of Related Work	159
5.2.1	Survey Criteria	159
5.2.2	Retrospective Changes to Established Blockchain Systems	159
5.2.3	Alternative Blockchain Designs with Pruning Capabilities	162
5.3	Design Goals	164
5.4	CoinPrune Overview	165
5.5	Retrofittable Block Pruning	167
5.5.1	Data Management	167
5.5.2	Coordination of Established Nodes	169
5.5.3	Bootstrapping of New Nodes	169

5.6	Handling Application-Level Data	170
5.6.1	Obfuscation of Illicit Data from the UTXO Set	170
5.6.2	Preservation of Application-Level Data	173
5.7	Seamless Integration into Bitcoin	174
5.7.1	Additional On-Chain Data	174
5.7.2	Adaptions to the Peer-to-Peer Protocol	174
5.8	Security Discussion	175
5.8.1	Snapshot Validity	175
5.8.2	Reliability of Reaffirmations	177
5.8.3	Peer-to-Peer Attacks	178
5.9	Performance Evaluation	179
5.9.1	Testbed Setup for Synchronization Measurements	179
5.9.2	Storage Savings	179
5.9.3	Evaluation of Synchronization Performance	181
5.10	Conclusion and Future Work	183
6	Blockchain-based Bootstrapping of Anonymity Services	185
6.1	Motivation	186
6.1.1	Problem Analysis	186
6.1.2	Contributions	190
6.2	Design Goals	191
6.3	AnonBoot Overview	192
6.3.1	Design Intuition	192
6.3.2	Overview of Bootstrapping Steps	193
6.4	Sybil-Resistant Index of Peers and Services	195
6.4.1	Message Types	195
6.4.2	Pulse-based Message Release	198
6.5	Bootstrapping Secure Anonymity Services	199
6.5.1	Connecting Users and Privacy Peers	200
6.5.2	Local Selection of Privacy Peers	200
6.5.3	Service Requests for Peer Election	201
6.6	Realization of Use Cases	202

6.6.1	Decentralized Onion Routing via AnonBoot	202
6.6.2	Shuffling Networks and Cryptocurrency Tumblers	203
6.6.3	User-Centered Redaction Juries	204
6.7	Security Discussion	205
6.7.1	Proof of Work Against Sybil Attacks	206
6.7.2	Security of Bootstrapped Services	207
6.8	Performance Evaluation	210
6.8.1	Synchronization with Host Blockchain	210
6.8.2	Small Blockchain Footprint and Low Costs	211
6.9	Related Work	212
6.10	Conclusion and Future Work	213
7	Conclusion	215
7.1	Contributions	216
7.1.1	Systematic Analysis of Non-Financial Blockchain Content . .	216
7.1.2	Mitigation of Unwanted Blockchain Content	216
7.1.3	Retrofitting Blockchains with Pruning Capabilities	217
7.1.4	Blockchain-based Bootstrapping of Anonymity Services	218
7.2	Future Work	219
7.3	Closing Remarks	220
	Abbreviations and Acronyms	221
	Bibliography	223

1

Introduction

In recent years, blockchains gained tremendous attention from academia, practitioners, and governmental stakeholders alike [Eu19]. The technology was pioneered by Bitcoin in 2008 [Nak08], where the blockchain was introduced as a backbone for creating a completely decentralized digital payment system, i.e., avoiding banks and other intermediaries in the payment process. More generally, Bitcoin-style blockchains constitute *distributed*, *publicly accessible*, and *immutable* append-only transaction ledgers that are maintained by a *permissionless* peer-to-peer (P2P) network where nodes can join and leave at will. Since anybody can obtain and validate all previous records, the blockchain provides a transaction history that all nodes agreed on. When Bob issues a transaction, e.g., transferring bitcoins to Alice, any node in the network would detect cheating attempts by him. Hence, Alice can now rely on the *network* to ensure the payment's correctness even if she does not fully trust Bob.

Subsequently, the potential of blockchains to facilitate interactions between mutually distrusting parties without having to rely on a trusted third party was seized for further applications beyond digital payment systems. Nowadays, blockchain systems improve interactions and processes in numerous domains, such as insurance industries [LCE18, BMW18, DTF19], education [GC17, THK⁺18, MPW20], healthcare [LZS⁺18, AFKU20, YSJA21], the Internet of Things [SBHD17, DKJ17, DZZ19], smart grids [MNB⁺18, GWZ⁺19, MZN⁺21], supply chain management [PHS18, JSK19, BPM⁺21], and business processes [MWA⁺18, DCCD⁺19, VXBS20]. These recent trends indicate a paradigm shift regarding the data that is recorded on the blockchain. While the transfer of bitcoins constitutes a self-contained blockchain application, i.e., transactions can be fully validated solely based on previous blockchain records, these new applications link external information sources to the blockchain. As a result, the data to be considered becomes more diverse and cannot be rigorously validated anymore before irrevocably being added to the blockchain [RKY⁺20]. The increased popularity and more data-intense use cases further aggravate the growing storage requirements of append-only blockchains as well.

These recent developments require that we establish a deeper understanding of the true benefits, but especially the limitations and challenges, of this *blockchain-based data management*. Namely, we need to investigate and control *what* and *how much* data gets irrevocably stored on a blockchain. The need for focusing on these two aspects is especially apparent considering permissionless blockchains, where anybody can operate a node and all nodes obtain and store a copy of all blockchain data.

We first need to investigate *what data* enters a blockchain, as node operators may face negative consequences as they store all blockchain content and distribute it to other nodes. Especially, sensitive information would be stored indefinitely and be obtainable for investigation by any interested user. For instance, previous studies analyzed the payments recorded on Bitcoin’s blockchain and showed that this approach can help deanonymize Bitcoin users [MPJ⁺13, RH13, OKH13, SMZ14, FKS15, MN22]. Data owners can exercise control over what data to publish on the blockchain, as they can cautiously vet the data before releasing it. In stark contrast to this controllable information disclosure, the users cannot influence what data *other* parties store on the shared blockchain as long as the validation process cannot reject the corresponding transactions upfront, as is the case for, e.g., attempts to double-spend bitcoins. Bitcoin transactions have been (mis)used in the past to encode other content than financial transactions, such as an image of Nelson Mandela or a copy of the original Bitcoin white paper [Shi14]. While such instances of *unwanted blockchain content* seem harmless, the full gravity of insufficient control over blockchain content unfolds when *illegal content* is irrevocably embedded: In early 2019, someone stored child abuse imagery on the blockchain of Bitcoin SV, which triggered police investigations in the UK [BBC19]. As all nodes, including all faithful nodes, would receive and, by default, store such illegal content, we must assess and mitigate such new challenges as part of the blockchain-based data management.

Furthermore, we have to mitigate the negative effects of *growing blockchain sizes*, as they may ultimately become prohibitive for large-scale and long-term utilization. Bitcoin’s blockchain already consumes hundreds of gigabytes of disk space [Blo11a]. As a result, synchronization times become burdening [CDE⁺16, Gen19, Lop20] and low-storage devices cannot operate a full Bitcoin node without, e.g., pruning blocks after processing them [Bit15b]. However, simply forgetting older blocks is detrimental to the network’s security, as other nodes rely on receiving their information from nodes that know the full blockchain [TS16]. Hence, blockchain-based data management further involves dealing with increasing blockchain sizes and investigating to which extent nodes can be relieved from prohibitive storage requirements.

These required considerations impose further limits on applications that can benefit from relying on blockchain technology to manage relevant data. Even though the technology has been praised for its potential by, e.g., German federal ministries [Bmwi18], a report published by the World Trade Organization (WTO) [Gan18], or the EU Commission [Eu19], reservations regarding the true benefits of blockchain technology persist [Sch19, Bar19]. These reservations further motivate the question of what applications naturally benefit from operating on top of a permissionless blockchain, i.e., to use the blockchain as a distributed medium for further coordination, especially in the light of the aggravated challenges regarding unwanted or illegal blockchain content and increasing blockchain sizes.

In this dissertation, we set out to adjust the promises of blockchain-based data management in the permissionless setting (a) by analyzing and addressing the novel challenges of unwanted blockchain content and increasing blockchain sizes, and (b) by investigating the utility of blockchain-based data management for novel applications. We further strive to maximize the applicability of our solutions by keeping their deployability to existing blockchain systems in mind. We first analyze the practice of storing unwanted content on a blockchain by assessing the available insertion methods, the dimensions of content insertion, and potential consequences for node operators, before proposing countermeasures to mitigate the negative impacts of such content. We then consider growing blockchain sizes and propose a gradually deployable protocol to drastically reduce the required storage for users of cryptocurrency blockchains such as Bitcoin. Finally, we propose a framework for securely bootstrapping distributed services that seizes the trust assumptions of a permissionless blockchain to illustrate a previously untapped potential for applications making use of a sustainable blockchain-based data management.

In the following, we first discuss the challenges stemming from relying on permissionless blockchains for a service's data management. From this analysis, we then derive research questions to help adjust the promises of blockchain-based data management. Finally, we give an overview of our contributions and outline the structure for this dissertation.

1.1 Problem Analysis

Blockchain-based data management promises to improve interactions between mutually distrusting peers beyond Bitcoin's initial scope of digital financial transactions. Blockchains provide a valuable building block for these settings due to their characteristic properties, which allow establishing decentralized, transparent, and immutable event ledgers. However, these features inherently create new problems for designing blockchain-based data-management systems.

In this section, we first give a brief overview of permissionless blockchains and their characteristic properties (Section 1.1.1). Based on this background, which we will extend upon in Chapter 2, we then discuss the problems permissionless blockchains introduce to the design of data-management systems (Section 1.1.2).

1.1.1 Promises of Blockchain Technology

Bitcoin introduced the blockchain as a decentralized, transparent, and immutable ledger for financial transactions to create a digital payment system without having to rely on a trusted third party, such as a bank. In the following, we give a brief overview of how permissionless blockchains operate in general, and we highlight characteristic properties of the technology and the opportunities these properties enable.

Blockchains are event ledgers that establish a globally shared state across the nodes of a P2P network. A blockchain consists of a list of cryptographically interlinked *blocks*, which in turn contain *transactions* previously proposed by users to update the shared state. For instance, Bitcoin transactions typically transfer bitcoins, the system’s currency, from one user to another. By including a transaction in the blockchain, the network reaches consensus about the transaction’s validity. In *permissionless* blockchains, anybody can run nodes on the network without further restriction, i.e., nodes can join and leave arbitrarily, and anybody can obtain, inspect, and revalidate the blockchain according to predefined rules [KAKC20]. Hence, blockchains can be used to establish consensus in a *decentralized* manner.

In the permissionless setting, dedicated nodes, called *miners*, compete to append the next block in exchange for a reward. This competition involves solving a computationally expensive *Proof of Work (PoW) puzzle*, which requires varying the block until its cryptographic hash value is sufficiently small [MKKS15]. Each block contains the hash value of its predecessor to chain the blocks. Because of this interlinking, the blockchain becomes virtually *immutable*: If someone now attempts to alter a block somewhere in the blockchain, then they have to recompute the computationally expensive PoW puzzle for all subsequent blocks as well.

Since the blockchain is a public and immutable ledger, it provides *public verifiability* and *transparency*: Once a transaction is recorded on the blockchain, that transaction cannot be revoked anymore due to the blockchain’s immutability. Furthermore, anybody can obtain and revalidate a full copy of the blockchain, i.e., the blockchain provides a complete history of past transactions for the public eye.

This transparency inherently clashes with privacy aspects [WG18]: Anybody can obtain and analyze a copy of the blockchain, e.g., to track users’ activities. Permissionless blockchains can provide *pseudonymity* [PK01], i.e., a user’s individual transactions may remain linkable for blockchain observers without inherently revealing the user’s identity [MPJ⁺13, OKH13, SMZ14, MN22]. However, users have to take further care to maintain this pseudonymity as additional side-channel information can help deanonymize a user’s pseudonyms [RH13, FKS15, HAL⁺18].

Finally, transactions contain instructions that nodes execute locally to transition from the previously agreed-upon state to a new shared state, i.e., blockchain systems rely on *script-based state updates*. While this concept was generalized and popularized by Ethereum [But13, Woo14], where transactions trigger functions defined in smart contracts that are distributed through the blockchain as well, even Bitcoin uses a stack-based scripting language for its transactions [Bit10d]. Script-based state updates ensure a certain flexibility to enable new applications, but the expressiveness of these scripts might be limited. For instance, Bitcoin only encourages the utilization of a few script templates (cf. Section 2.3.3.3) and removed certain previously defined operations entirely for security reasons [Bit10d].

In summary, the characteristic properties we outlined in this section ensure that permissionless blockchains can provide a distributed and tamper-resistant event ledger, which facilitates public verifiability. With these properties, blockchains promise to provide a cryptographically secured and decentralized trust mediator for interactions

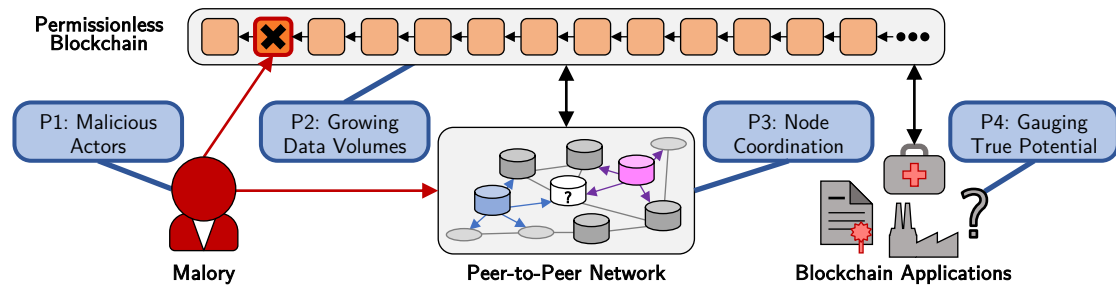


Figure 1.1 Blockchain-based data management must account for malicious actors as well as growing data volumes due to the blockchain’s append-only nature. Addressing these concerns further requires dedicated coordination among the nodes of the underlying P2P network. Finally, fully seizing the potential of blockchain-based data management remains challenging.

between users who do not fully trust each other [WG18] and to potentially improve accountability due to its public verifiability [Gan18, Eu19]. These promises have led to a widespread application of blockchain technology in numerous use cases, such as insurance, healthcare, education, or supply chain management. However, the characteristic properties enabling this innovation come with inherent new problems considering data-management aspects, as we discuss in the next section.

1.1.2 Problems of Blockchain-based Data Management

We now assess the new problems for designing data-management systems based on permissionless blockchains that inherently stem from these blockchains’ characteristic properties (cf. Section 1.1.1). Namely, we identify the following four core problems as illustrated by Figure 1.1: The presence of *malicious actors*, *ever-growing data volumes*, the need for decentralized *node coordination*, and successfully *gauging the true potential* of permissionless blockchains to fuel decentralized applications.

P1: Malicious Actors

Permissionless blockchain systems [KAKC20] are especially subjected to *malicious actors* as their openness, pseudonymity, and the financial values involved incentivize adversarial or even criminal behavior. In the past, blockchain-based cryptocurrencies have been linked to, among others, theft [MPJ⁺13], money laundering [MBB13], and ransomware [HAL⁺18]. In fact, Bitcoin’s main promise was to solve the double-spending problem, i.e., mitigating the possibility to duplicate digital coins, in decentralized settings without the need for a trusted third party [Nak08]. Other threats for blockchain-based currencies [TS16] engulf, e.g., attacks on the fairness of the mining process [ES14], denial-of-service (DoS) attacks [MJP⁺20], or further attacks on the users’ privacy, such as hindering the Bitcoin network’s accessibility through the anonymity network Tor [BP15].

The transition to a generalized blockchain-based data management creates additional threats regarding malicious actors. First, managing data from external sources via the blockchain has the inherent risk that users submit invalid data due to a lack

of verifiability of such information [RKY⁺20]. Second, malicious actors could irrevocably publish sensitive data, such as someone’s private information, intellectual property, or classified documents, via the blockchain. In the past [Shi14], Bitcoin has been (ab)used to store, for instance, personal photographs, leaked private keys, and a copy of Wikileaks’ “Cablegate” [Lyn13] documents. Third, beside this arguably *unwanted* blockchain data, malicious actors could also add *illegal* content to the blockchain, which would then be obtained, stored, and redistributed by other, honest nodes of the blockchain network. The incident of child abuse imagery on Bitcoin SV’s blockchain in 2019 [BBC19], which immediately followed a change allowing storing larger files more easily on the blockchain [Mon19b, Mon19a], has to serve as an alarming example in this regard. While others have highlighted potential harm stemming from the insertion of illegal content prior to the incident [AMVA17, MLS⁺15, PDC17], the true gravity of such adversarial behavior remains to be investigated.

P2: Increasing Data Volumes

Permissionless blockchains rely on the presence of sufficiently many *full nodes* on the P2P network, i.e., nodes that maintain a full local copy of the blockchain. Newly joining nodes only know a hard-coded genesis block, and they have to obtain all blocks from different full nodes during their initial synchronization process. The joining nodes can only trust the blocks they receive when there are sufficiently diverse and reliable sources to obtain the blockchain.

However, the immutability property causes blockchains to be *ever-growing*, i.e., their size is going to grow continually without taking further measures. This growth becomes increasingly challenging because every full node should keep a full copy of the blockchain to maintain a desirable level of redundancy. For instance, Bitcoin’s blockchain has a size of 416 GiB as of Jan 4, 2023 [Blo11a]. With its current average growth rate of 152 MiB/d [Blo11a], Bitcoin will reach a milestone around Jul 24, 2024 where nodes will have to reserve 500 GiB of storage only to store the blockchain.

Therefore, these ever-growing sizes constitute a severe problem for blockchain-based data management. The desired massive duplication of all blockchain data aggravates this issue, as it affects each full node individually. Conversely, prohibitive blockchain sizes incentivize node behavior that is detrimental to the overall network’s best interests, such as pruning older blocks [Bit15b]. Hence, positively contributing to the network becomes ultimately irrational for individual nodes as long as the problem of growing blockchain sizes remains unaddressed.

P3: Coordination and Updatability

When attempting to address problems of blockchain systems, such as mitigating new adversarial behavior by malicious actors or counteracting increasing data volumes, system designers face the additional problem of required *node coordination*: Since the blockchain is maintained by a decentralized P2P network, all nodes have to agree on enforcing the same set of policies.

This problem is further aggravated whenever addressing an issue requires *updating* the policies of a live blockchain network. Ideally, all nodes update their policies at the same time. In reality, however, every node is free to choose if and when to update. These uncertainties create a potential for blockchain *forks*, i.e., different nodes of the same network accept or reject different information inconsistently and thereby cause a split of the blockchain. For instance, a bug fix affecting which blocks were accepted by Bitcoin nodes created an unintended fork of Bitcoin’s blockchain in March 2013 as some nodes upgraded to include the bug fix and others did not [And13, BMC⁺15]. Larger mining pools downgraded to the older Bitcoin client to resolve this fork as early as possible, but thereby knowingly disregarded the bug fix [And13]. While the issue was ultimately resolved [BMC⁺15], this incident illustrates that required updates must be deployed with care to ensure the robustness of the system.

Hence, the coordination among blockchain nodes remains challenging, especially considering the post-deployment updates required to address newly identified issues with the shift to a generalized blockchain-based data management.

P4: Gauging the True Potential for Applications

Following the success of Bitcoin as a cryptocurrency and Ethereum as a smart-contract platform, blockchain technology was praised as a “building block for the [I]nternet of the future” [Bmwi18]. In the same vein, a report published by the WTO anticipated that “[blockchain] could be to transactions what the [I]nternet was to communication” [Gan18] and the EU Commission acknowledged that “a vast set of opportunities can emerge from blockchain as a technology” [Eu19].

However, there are reservations regarding the true potential of blockchains as a building block for data-management systems. Arguments comprise, for instance, that blockchain technology merely transformed a very specific sub-category of “trust” [Sch19] or that using permissionless blockchains in the envisioned use cases proves much more challenging than previously anticipated [Bar19].

A final problem of blockchain-based data management is, thus, a rather fundamental one: What are the use cases that require applications based on permissionless blockchains? While general guidelines emerged that attempt to answer when service operators can sensibly use a blockchain [WG18], we currently lack insights into which applications are enabled or allow for new desirable features by operating on top of permissionless blockchains.

Summary. In conclusion, blockchain technology promises to mediate trust between mutually distrusting parties without the need to involve an especially trusted third party. Instead, the parties involved can trust the decentralized, immutable, and transparent public ledger that is established via the blockchain. However, the shift to using blockchains for other data than financial transactions, which can be validated intrinsically based on the blockchain’s history, and a resulting increased popularity of the technology unveils new problems. In the next section, we pose research challenges for successfully adjusting the promises of this generalized blockchain-based data management.

1.2 Research Questions

Our problem analysis in Section 1.1 unveiled a dichotomy of the utility of permissionless blockchains: While these blockchains offer distinctive and desirable properties that, on paper, benefit the design of decentralized services, these properties introduce new problems in the context of blockchain-based data management and ultimately limit the technology’s true applicability.

Thus, these problems must be addressed to unfold the full potential of permissionless blockchains and to ensure a flourishing and sustainable infrastructure for modern distributed services. In this dissertation, we aim to adjust the promises of blockchain-based data management in the permissionless setting by tackling these problems. In this vein, our main goal is to establish a deeper understanding of the implications of blockchain-based data management, investigate technical requirements to overcome the aforementioned challenges, and explore untapped potentials.

Accordingly, we define three research questions, which we derive from our problem analysis, to guide the contributions of this dissertation. First, we investigate how we can *moderate* what data is persisted on the blockchain to mitigate the potential for unwanted or even illegal content as well as the ever-growing sizes. Second, we set out to determine which corresponding changes can, and cannot, be deployed by *retrofitting* existing blockchain systems, and thereby maintain the levels of trust such systems already gained, instead of designing new systems from scratch. Finally, we search for *novel applications* that inherently benefit from relying on permissionless blockchains despite the identified limitations.

Q1: How can we moderate the content stored on permissionless blockchains?

Immutability and public verifiability of data records are desirable properties of permissionless blockchains, as we established in Section 1.1.1. However, we also discussed the corresponding problems that arise regarding the blockchain-based data management in Section 1.1.2, i.e., malicious actors being able to irrevocably store unwanted or even illegal data (**P1**) and ever-growing data volumes (**P2**). Since these problems imply risks and burdens for the operators of blockchain nodes, solutions for limiting what and how much data can be persisted in the blockchain are required. Hence, we need to further understand the phenomena of blockchain content insertion and ever-growing blockchain sizes and investigate to which extent we can moderate or otherwise steer a blockchain’s content to mitigate the risks for node operators.

Q2: Can we retrofit existing blockchain systems to address arising challenges instead of deploying new systems?

Addressing any problem of permissionless blockchains, such as developing a moderation process (**Q1**) for controlling blockchain content (**P1**) and sizes (**P2**), requires thorough deployability considerations. Even though new blockchain designs addressing individual problems have been proposed frequently (e.g., [Bru14, Poe16,

BKLZ20]), these systems have to compete against the more popular long-running blockchain systems [SA21]. For instance, Bitcoin and Ethereum have a combined dominance of roughly 60% in the cryptocurrency market [Coi13]. Because of this popularity, these established systems are especially prone to malicious actors (**P1**) and large blockchain sizes (**P2**). When addressing these problems, already established blockchain systems thus require special attention. Accordingly, we have to ask whether any proposed changes can be retrofitted to existing blockchain systems with minimal impact on node coordination (**P3**), or whether there are solutions that warrant substantial changes to prior blockchain designs.

Q3: How do permissionless blockchains promote novel applications despite their data-management limitations?

While blockchain technology has been praised for its potential, we have also highlighted reservations that point out limitations of the technology’s utility for decentralized applications (cf. Section 1.1.2). To address our final problem, we thus ask whether permissionless blockchains can be used as a building block to realize novel applications that inherently benefit from the blockchain’s characteristic properties (**P4**). Additionally, we strive for a non-invasive integration of an existing blockchain to leverage its established trustworthiness without overburdening it.

These research questions guide our path to demystifying and adjusting the promises of blockchain-based data management in the permissionless setting. In this dissertation, we answer these research questions by proposing technical solutions that address the identified challenges and seize the potential of blockchains while explicitly considering their applicability to already established blockchains. Our research therefore focuses on Bitcoin and Bitcoin-like blockchains, which comes with additional benefits. First, the identified challenges are already pressing for Bitcoin due to its sustained popularity, i.e., Bitcoin is in immediate need of respective remedies. Second, overcoming Bitcoin’s comparably limited programmability and extensibility provides paths to apply corresponding solutions to a wide array of blockchain systems. Finally, novel applications that can operate on top of Bitcoin without interfering notably with its normal operation hint at sensible data-management solutions.

1.3 Contributions

In this dissertation, we present *four* distinct contributions (**C1–C4**) to address our research questions (**Q1–Q3**). Figure 1.2 shows both how our contributions map to the postulated research questions and to which extent the identified problems for a blockchain-based data management (**P1–P4**) influence the respective contributions. Briefly summarized, our contributions are as follows:

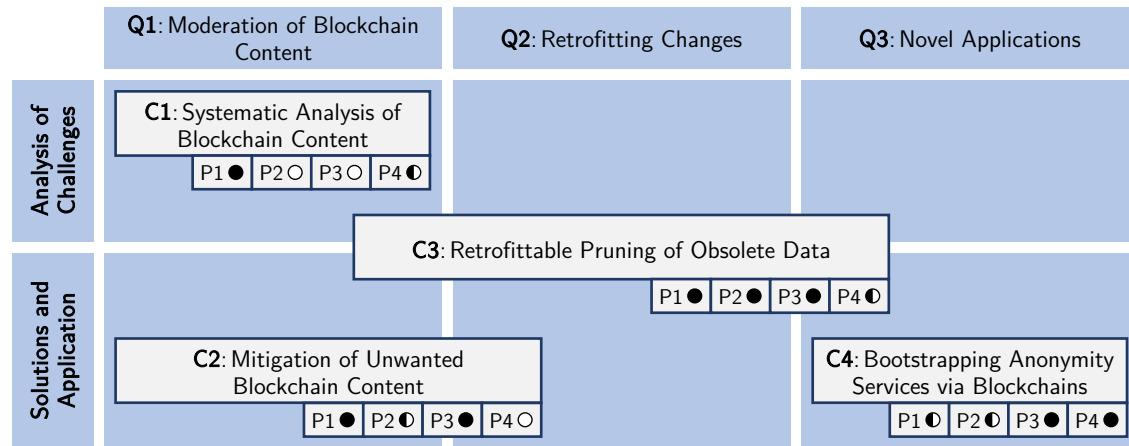


Figure 1.2 Relation of our contributions (**C1–C4**) to the defined research questions (**Q1–Q3**). We further highlight whether the problems of blockchain-based data management (**P1–P4**) influence the respective contributions strongly (●), moderately (◐), or not at all (○).

- C1:** We first quantify and assess the presence of non-financial data on the blockchains of Bitcoin and related forks to highlight potential negative consequences of unmoderated content insertion for widely used permissionless blockchains.
- C2:** We then address the issue of unwanted content insertion for permissionless blockchains by discussing options to *mitigate* as well as *retroactively remove* such content while discussing the retroactive deployability to Bitcoin.
- C3:** We enable Bitcoin nodes to prune obsolete data from their blockchain copy without impeding the overall network’s capability to bootstrap new nodes. Our pruning scheme is gradually deployable and additionally obfuscates unwanted content while preserving other application data from third-party services.
- C4:** Finally, we present a platform for bootstrapping distributed anonymity services as a novel application that is enabled by blockchain-based data management. Namely, the platform’s users coordinate solely via Bitcoin as a trust anchor.

With these contributions, we first analyze the problems of blockchain-based data management in the context of Bitcoin and then propose technical solutions to address these problems. Namely, Contributions **C1** and **C2** investigate the need for, and the possibilities of, moderating the content stored on permissionless blockchains, whereas Contribution **C3** addresses these blockchains’ pressing issue of ever-growing sizes. Finally, Contribution **C4** demonstrates how the characteristic properties of permissionless blockchains can be seized to realize novel applications.

C1: Systematic Analysis of Non-Financial Blockchain Content

Even before transitioning to generalized blockchain-based data management, users have irrevocably engraved files into Bitcoin’s blockchain [Shi14]. While enabling numerous applications (**P4**), this shift also lowers the bar for malicious actors to engrave unwanted or illegal content (**P1**), which would then be distributed across the underlying P2P network. Hence, this phenomenon requires further investigation.

In our first contribution, we systematically analyze the inclusion of unwanted blockchain content on the blockchains of Bitcoin and related forks from three perspectives. First, we analyze the technical options available to users to insert content other than financial transactions into Bitcoin’s blockchain (Section 3.3). Second, we assess potential consequences of inserting such content, both on a technical level and regarding legal consequences for operating a Bitcoin node (Section 3.4). Third, we develop content detectors to identify and quantify transactions that use the respective content insertion methods (Section 3.5). Individual findings of non-financial blockchain content [Shi14] and potential consequences [AMVA17, MLS⁺15, PDC17] have been discussed prior to our work. However, to the best of our knowledge, our work constituted the first systematic analysis of this phenomenon.

Our analyses indicate that Bitcoin users regularly make use of unintended methods for inserting blockchain content. Even though the overall utilization of these methods remains comparably low, our analysis unveils that over 1600 accessible files had been irrevocably stored on Bitcoin’s blockchain up to August 2017. Among these files, we identified individual instances of objectionable and potentially illegal content. We subsequently extended our analysis and considered more Bitcoin-related forks as well as new blocks mined since our initial investigation. Namely, our updated analysis additionally covers the blockchains of Bitcoin Cash and Bitcoin SV, and we now consider blocks up to July 2020 in all three blockchains. However, we refrain from assessing new content in detail, again due to the discovery of child abuse imagery on Bitcoin SV’s blockchain in 2019 [BBC19]. Nevertheless, this updated analysis indicates a distinctive shift regarding content insertion, as Bitcoin SV allows for the intended insertion of much larger data volumes by using the Bitcoin Data Protocol. As a consequence, the utilization of traditional insertion methods declined and is superseded by the Bitcoin Data Protocol on Bitcoin SV’s blockchain, which is used extensively compared to the historical utilization of traditional methods. Namely, we now observe tens of thousands of (alleged) files being irrevocably engraved this way per month. These observations underpin the urgency of addressing the question of how to enable the oversight and moderation of blockchain content (**Q1**), such that negative consequences for honest blockchain users can be mitigated.

C2: Mitigation of Unwanted Blockchain Content

After having identified (**C1**) a need for moderating the content stored on permissionless blockchains (**Q1**), we present measures to mitigate unwanted content insertion in our next contribution. While we show that content insertion cannot be prevented entirely even for simple blockchains such as Bitcoin (Section 4.1), our countermeasures increase the resilience against such content.

First, in Section 4.2, we explore the design space for *preventing* content insertion. Namely, we assess the efficiency and deployability (**Q2**) of (a) explicitly filtering for content as part of the block validation rules similarly to how virus scanners protect computing devices today, (b) increasing the costs of large transactions to prohibitive levels through mandatory minimum fees, and (c) hardening Bitcoin transactions against manipulation by relying on one-way commitments instead of easily manipulable identifiers. Our results show that these mitigation strategies can complement

each other, but mandatory minimum fees promise the most robust deterrent for content insertion. Contrarily, content filters can hardly be kept up to date in a massively distributed system relying on global consensus (**P3**) and one-way commitments imply considerable storage overheads (**P2**).

Second, in Section 4.3, we propose *RedactChain*, a moderation framework for permissionless blockchains that enables nodes to *actively* revert content insertion in a swift, yet trustworthy manner. RedactChain achieves this goal by periodically electing small redaction juries that can concurrently remove illicit content from transactions when reported by users. By enabling these redaction juries to only jointly alter a short sequence of blocks without breaking the blockchain’s integrity according to a restrictive set of rules, the jury members’ control remains limited, and we can effectively prevent misuse by malicious actors (**P1**). Furthermore, all executed blockchain alterations are recorded transparently on an additional redaction log. Hence, we ensure that all nodes can properly coordinate (**P3**) in face of required alterations. Our results show that blockchain moderation can be realized with a manageable storage overhead compared to non-redactable blockchains. Depending on the jury size, redactions can take from 10s to 1 min for smaller juries (16 to 28 members), but times increase further for larger juries (e.g., 14 min for 58 members). The additional coordination inflicts a manageable overhead (**P2**) below 1.5 kB per redaction and below 3.3 kB per block, which would reduce the capacity of a block by only 15 transactions on average when compared to Bitcoin. While our prevention strategies remain deployable to Bitcoin with only marginal changes, RedactChain illustrates that additional moderation capabilities can be unveiled when allowing for more liberal design choices based on Bitcoin’s initial model.

C3: Retrofitting Blockchains with Pruning Capabilities

Besides the potential negative impact of unwanted blockchain content, we identified the ever-growing storage requirements (**P2**) as the second major problem of blockchain-based data management. This problem is especially aggravated if blockchains experience frequent transactions, e.g., due to a large popularity of services provided on top of that blockchain, and long-term utilization. At the same time, however, deploying required changes is particularly hard to coordinate in such popular systems (**P3**). In summary, ever-growing blockchain sizes especially require retrofittable solutions to be able to react to the problem post-deployment (**Q2**).

As our third contribution, we thus present *CoinPrune*, a secure block-pruning protocol for Bitcoin and similar permissionless blockchains that is retrofittable via a velvet fork [KMZ20, ZSJ⁺19]. CoinPrune unburdens the full nodes from having to maintain a complete copy of the Bitcoin blockchain, which is required to help new nodes synchronize, by enabling a synchronization based on recent snapshots of the relevant state instead. CoinPrune nodes periodically create new snapshots, and the miners *mutually reaffirm* the correctness of these snapshots on Bitcoin’s blockchain. This way, joining nodes can assess the validity of the snapshot they received before accepting it. Besides reducing the storage requirements for all nodes, CoinPrune’s snapshot creation is capable of additionally obfuscating unwanted blockchain content

that is not prunable otherwise (**P1**, **Q1**), and we can preserve benign application data during the pruning operation to prevent breaking Bitcoin-based services (**P4**, **Q3**).

Our evaluation shows that we can reduce the storage requirements for nodes from 312 GiB to 10 GiB compared to a vanilla Bitcoin client (for the first 640 000 blocks). This reduction directly translates to less network traffic and lower verification costs during the initial synchronization. Hence, during our measurements, we were able to reduce the time required for bootstrapping a new node from 7 h to 51 min.

C4: Blockchain-based Bootstrapping of Anonymity Services

Even though the advent of blockchain technology has directly impacted established business processes from various domains and also enabled entirely new applications, such as digital notary services or the fair exchange of digital goods, there are reservations regarding the limitations of permissionless blockchains (**P4**).

Our final contribution therefore addresses our research question how these blockchains can promote novel applications despite these limitations (**Q3**). To this end, we present *AnonBoot*, a decentralized platform for bootstrapping distributed anonymity services that utilizes a permissionless *host blockchain* for a periodic coordination between its users (**P3**). Specifically, AnonBoot maintains a Sybil-resistant repository of peers willing to contribute to providing anonymity services. AnonBoot uses Bitcoin’s blockchain to coordinate these independent peers and thereby benefits from the characteristic blockchain properties (cf. Section 1.1.1). This way, AnonBoot (a) provides a decentralized alternative for the directory services of anonymity networks such as Tor [DMS04] and (b) coordinates the bootstrapping of persisted services, e.g., mixnet-based cryptocurrency tumblers, such as CoinParty networks [ZGH⁺15, ZMH⁺18]. To the best of our knowledge, AnonBoot is the first fully decentralized solution to bootstrapping such anonymity services in a Sybil-resistant manner, i.e., no malicious actor (**P1**) can easily trick users into trusting a compromised service under his control.

Our evaluation shows that AnonBoot can operate on top of Bitcoin using only the restricted intended content insertion methods (cf. Section 3.3.1) and only periodically requires writing new on-chain messages. Hence, AnonBoot utilizes Bitcoin responsibly (**P2**), e.g., AnonBoot can maintain a repository of 1000 privacy peers and bootstrap over 100 mixnet-based anonymity services even when only consuming up to 5% of Bitcoin’s block size during a 5-block coordination phase.

Interrelation of Contributions

We identified the irrevocable insertion of unwanted or illegal content (**P1**) and increasing data volumes (**P2**) as the main problems of blockchain-based data management, which are further aggravated in the permissionless setting due to the increased complexity of coordinating and updating nodes of the underlying P2P network (**P3**). In this dissertation, we first analyze the impact of these problems on the operation in the permissionless blockchains in Contribution **C1** and as part of

Contribution **C3**, respectively. We then propose solutions to both problems via our Contributions **C2** and **C3**. Both contributions constitute steps toward the much-needed moderation (**Q1**) of blockchain contents and data volumes, while keeping the deployability of our solutions a focus of our considerations (**Q2**). With Contribution **C4**, we then illustrate that a blockchain-based data management can enable novel applications (**Q3**) despite its recently identified limitations (**P4**).

1.3.1 Attribution of Contributions

Most contributions we present in this dissertation have been created with the support of students at the Chair of Communication and Distributed Systems at RWTH Aachen University (COMSYS) either in the context of their Bachelor's and Master's theses or student assistant positions supervised by the author of this dissertation. The resulting publications, which provide the basis for this dissertation, were created with the support of the publications' respective co-authors. In the following, we attribute the individual involvement of the students and co-authors to the respective final contribution. If not stated otherwise, the responsibility for the initial ideas and concepts, the designs of the corresponding solutions, the concepts for required evaluations and measurements, and drafting the initial write-up and the final publication of results lies with the author of this dissertation.

Contribution **C1** consists of three stages. The initial suggestion to work on blockchain content was made by Jan Henrik Ziegeldorf upon discovering a content insertion service for Bitcoin (cf. Section 3.3.2). The author of this dissertation and Jens Hiller then explored and discussed that service and realized a potential for negative consequences of blockchain data storage. A first feasibility study on the non-financial content on Bitcoin's blockchain was based on the measurements Robin Rawiel conducted in his Bachelor's thesis [Raw16], which was supervised by the author of this dissertation. Oliver Hohlfeld contributed to the data analysis for the subsequent poster abstract [MHH⁺16], especially by analyzing URLs obtained from Bitcoin transactions. The writing process was supported by Martin Henze and Oliver Hohlfeld, and Jan Henrik Ziegeldorf further revised the paper. As a basis for the first full content analysis, the author of this dissertation reimplemented and extended Robin Rawiel's initial measurement framework. The author of this dissertation discussed the findings at multiple stages with Jens Hiller, Oliver Hohlfeld, and Martin Henze, who provided their insights to support the measurements. Finally, Dirk Müllmann provided the analysis of potential legal consequences for Bitcoin node operators in Germany based on technical details provided by the author of this dissertation. All co-authors then revised the final publication [MHH⁺18]. The updated study was supported by Muhammad Aitsam during his internship with COMSYS and by Roman Karwacik as part of his student assistant position. Roman Karwacik generalized the measurement framework and gathered and preprocessed updated data from the blockchains of Bitcoin Core, Bitcoin Cash, and Bitcoin SV. The author of this dissertation then drafted a book chapter [MHMW24] that incorporated these updated results as well as an updated discussion of legal consequences by Dirk Müllmann and that was then revised by Martin Henze.

Contribution **C2** engulfs our mitigation and moderation strategies against unwanted blockchain content. The value of prevention strategies against unwanted blockchain content, i.e., the first part of Contribution **C2**, was first brought up by Martin Henze during discussions of our content analysis [MHH⁺18]. The author of this dissertation then developed the mitigation strategies and discussed them with all co-authors. The corresponding publication [MHZ⁺18] was then initially drafted by the author of this dissertation and revised by all co-authors. When studying existing approaches to create redactable blockchains, the author of this dissertation then identified the need for swift and transparent redactions and conceptualized RedactChain, i.e., the second part of Contribution **C2**. Vincent Ahlrichs then created an initial prototype of RedactChain in his Master’s thesis [Ahl20]. The initial design evolved subsequently based on discussions between the author of this dissertation, Vincent Ahlrichs, and Jan Pennekamp. Roman Karwacik extended the initial prototype as a student assistant, and he further conducted an initial performance evaluation of this prototype. The author of this dissertation subsequently reimplemented and extended the prototype as a basis for the performance evaluation and proceeded to draft a paper [MAP⁺22a], which was then revised by Jan Pennekamp.

Contribution **C3** consists of our block-pruning scheme CoinPrune. Based on an idea and design by the author of this dissertation, which was refined based on discussions with Martin Henze, Benedikt Kalde created the first CoinPrune prototype during his Master’s thesis [Kal17]. This thesis was mainly supervised by the author of this dissertation, and Felix Konstantin Maurer contributed further ideas for smaller technical improvements of the design as a co-supervisor. Arthur Drichel improved the initial prototype created by Benedikt Kalde to parallelize the initial snapshot acquisition of joining CoinPrune nodes. Martin Henze additionally suggested that CoinPrune nodes can also obtain snapshots from third parties and not only from the Bitcoin network. The author of this dissertation then further refactored the code and based it on a newer version of the vanilla Bitcoin client, and he integrated various smaller improvements for increased robustness. The author of this dissertation then drafted the initial draft of our preliminary report on CoinPrune [MKP⁺20a], which was then revised by Martin Henze and Jan Pennekamp. The author of this dissertation further conceptualized the extensions of CoinPrune presented in the full paper [MKP⁺21], which was again revised by Martin Henze and Jan Pennekamp.

Contribution **C4** consists of our Bitcoin-based bootstrapping platform for anonymity services, AnonBoot. The author of this dissertation initially considered bootstrapping mixnet-based cryptotumblers via a distributed hash table (DHT). A prototype for this DHT-based platform was developed by Alexander Theißen in his Master’s thesis [The17]. The author of this dissertation then decided to shift development from a DHT-based platform to use Bitcoin as a foundation in joint work with Jan Pennekamp. Jöran Wiechert helped improve the design of AnonBoot and developed the majority of our prototype’s code base in his Bachelor’s thesis, which was ultimately not submitted. Erik Buchholz then created the first full version of the prototype as part of his student assistant position, which was then partially refactored by the author of this dissertation. Erik Buchholz also orchestrated the prototype’s performance evaluation. The author of this dissertation then drafted the corresponding publication [MPBW20a], which was then revised by all co-authors.

1.4 Outline

The main part of this dissertation is organized as follows. Chapter 2 provides an introduction to blockchain technology and Bitcoin as a foundation for our contributions. In the subsequent chapters, we present the individual contributions of this dissertation. In Chapters 3 and 4, we analyze the nature and extent of unwanted blockchain content and discuss corresponding remedies. First, in Chapter 3, we investigate the methods available to insert non-financial content into Bitcoin's blockchain, we quantify to which extent these methods have been utilized in the past, and we discuss potential consequences (**C1**). Then, we present our mitigation strategies against unwanted content insertion (**C2**) in Chapter 4. Afterward, Chapter 5 considers the increasing data volumes of permissionless blockchains. We discuss the negative impacts of ever-growing blockchains and present our block-pruning scheme that is retrofittable to Bitcoin as a technical solution to these problems (**C3**). In Chapter 6, we then present our final contribution, which focuses on gauging the true benefits of blockchain-based data management for other applications. Namely, we present our Sybil-resistant bootstrapping platform for distributed anonymity services that relies on Bitcoin as its host blockchain (**C4**). Chapter 7 then concludes this dissertation. In this final chapter, we summarize our contributions, formulate our key insights, and discuss directions for future work.

2

Bitcoin and Blockchain Technology

In this chapter, we provide an overview of Bitcoin [Nak08], its components, processes, and data structures. With this overview, we establish the relevant technical background for permissionless blockchains in general, and Bitcoin in particular, to serve as a basis for the contributions described in this dissertation. These contributions directly analyze or extend Bitcoin, respectively; however, one goal for our contributions is that they also remain adaptable to other blockchain systems. Hence, we do not only present all relevant technical details of Bitcoin, but briefly relate Bitcoin to other blockchain systems as well.

We largely base our overview of Bitcoin on various and historically scattered sources: While the idea of Bitcoin was given in Nakamoto’s white paper [Nak08], the technical documentation of Bitcoin is mostly maintained in the Bitcoin Wiki.¹ Additionally, consensus-critical updates to Bitcoin, i.e., updates that affect all nodes of the Bitcoin network, are coordinated via Bitcoin Improvement Proposals (BIPs). In our overview, we augment these primary sources with (technical) overviews of Bitcoin, e.g., the survey papers by Bonneau et al. [BMC⁺15] and Tschorsch and Scheuermann [TS16], or the books by Antonopoulos [Ano17] and Song [Son19]. Finally, we also consider the source code of Bitcoin Core,² the de facto reference implementation of Bitcoin. We base the overview in this chapter on the recent Bitcoin Core v22.0 [Bit22], but the contributions of this dissertation are historically affected by older versions of Bitcoin Core as well. Hence, we will highlight version differences as necessary in the remainder of this dissertation.

We start this chapter with a high-level overview of Bitcoin (Section 2.1), which also outlines characteristic properties of its blockchain and relates Bitcoin to other blockchain systems. Afterward, we reiterate Bitcoin’s underlying data structures.

¹<https://en.bitcoin.it>

²<https://bitcoin.org>

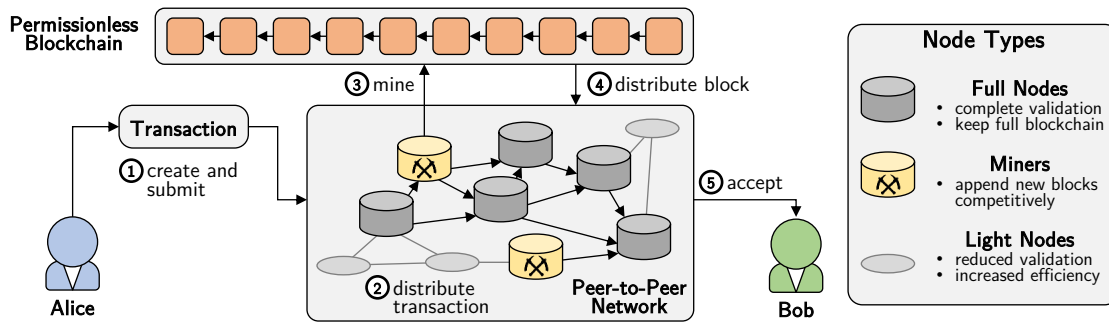


Figure 2.1 The blockchain is the core of Bitcoin’s design and establishes a medium of consensus for all participants, which is jointly maintained by a peer-to-peer network. To transfer some of his bitcoins to Bob, Alice ① creates a transaction and submits it to the Bitcoin network. The transaction is then ② distributed to all nodes of the Bitcoin network, where it is validated by the full nodes, before miners attempt to include it into a block as part of the ③ mining process. Once a miner successfully included the transaction into a block, that block gets ④ distributed as well and the nodes append it to their local blockchain copy after validating it. Finally, Bob ⑤ accepts that he correctly received Alice’s payment once the block containing his transaction was confirmed by at least six other blocks.

Namely, we first detail how Bitcoin’s blockchain and its blocks are structured (Section 2.2) before presenting Bitcoin’s script-based management of transactions, i.e., how users can transfer their coins to each other (Section 2.3). We then give a high-level introduction to Bitcoin’s consensus protocol by presenting the mining process, how other users validate new blocks and transactions, and how the nodes deal with concurrency (Section 2.4). Next, we give an overview of the operation of Bitcoin’s peer-to-peer network (Section 2.5), where we consider the information exchange between nodes and the initial synchronization process for joining nodes. Finally, we conclude this chapter with a brief summary (Section 2.6).

2.1 Bitcoin Overview

Bitcoin is a digital and decentralized payment system building upon cryptographic guarantees, commonly referred to as a *cryptocurrency*, which was introduced in 2008 by pseudonymous author Satoshi Nakamoto [Nak08]. At the core of Bitcoin’s design lies the *blockchain*, a distributed ledger of financial transactions that is publicly accessible but hard to tamper with. The main purpose of the blockchain is to keep track of the distribution of the system’s currency, the *bitcoin (BTC)*, which can be subdivided into *satoshis (Sat)*, where $1 \text{ BTC} = 10^8 \text{ Sat}$. With its blockchain, Bitcoin pioneered what would evolve into *blockchain technology* [YMRS18] and maintains a considerable dominance within the overall cryptocurrency market [Coi13].

Figure 2.1 gives an overview of the operation of Bitcoin that provides a basic model for other blockchain-based systems as well. Most fundamentally, Bitcoin offers its users the possibility to trade bitcoins without having to rely on a dedicated third party. This trade system is operated by a decentralized and *permissionless* peer-to-peer (P2P) network, i.e., nodes can join or leave at will without special restrictions.

The nodes of this P2P network, which we also refer to as the *Bitcoin network* in this dissertation, jointly validate the correctness of payments and append valid payments to the blockchain in an immutable manner. Immutability is achieved by the way the blockchain is constructed: The payments are bundled in *blocks* that are cryptographically chained, thereby forming the blockchain. More specifically, each block contains a cryptographic hash value of its predecessor, and appending a new block requires dedicated nodes of the Bitcoin network to solve a computationally hard puzzle. Attempting to alter a block somewhere within the chain requires solving that puzzle again. However, this new solution would not match the hash value stored in the block's successor anymore. As a consequence, the successor also has to be recomputed and the modification ripples on for all following blocks, requiring to solve the puzzle again for all affected blocks to prevent breaking the chain. Once a payment is recorded on the blockchain, that payment thus becomes virtually immutable after a couple of additional blocks are added to the blockchain afterward. In that case, the nodes agree that the payment was successful, and they thereby establish a network-wide *consensus* about what payments occurred in the past.

As Figure 2.1 illustrates, Bitcoin users and the Bitcoin network establish consensus about a new payment, in our example from Alice to Bob, in five steps as follows.

Alice first creates a *transaction* expressing her payment to Bob and submits it to the Bitcoin network (Step ①). To do so, Alice has to convince the Bitcoin network that she previously obtained the bitcoins she intends to spend, and she needs to specify Bob as the designated recipient. For this purpose, each transaction consists of *inputs* and *outputs*, where inputs unlock previously acquired bitcoins and outputs transfer those bitcoins to the new owner. Coin ownership is typically handled via public-key cryptography: Before Bob can accept payments, he has to create a public-private key pair first. He derives a *Bitcoin address* from his public key, which he can share with Alice. Alice then references Bob's Bitcoin address in the output of her transaction, and she submits her transaction to the Bitcoin network in order to send Bob her bitcoins. Bob can later spend these bitcoins again by creating a new transaction, referencing Alice's transaction output in an input of his new transaction, and proving that he controls the respective Bitcoin address by creating a signature using his corresponding private key. Bitcoin relies on a stack-based scripting language to express how the bitcoins associated with a transaction output can be spent later and to programmatically satisfy this condition in a future transaction's input. We will further detail Bitcoin's transaction system and its management of coin ownership in Section 2.3. In reality, Alice and Bob do not have to tediously manage their keys and Bitcoin addresses themselves. They rather use a *wallet* software, which provides a user interface that is reminiscent of online banking software and manages Bitcoin addresses, the corresponding keys, and available funds accordingly [Ano17].

Before Bob can further spend Alice's bitcoins, however, he must first be ensured that Alice's payment is valid, i.e., her transaction rightfully transfers bitcoins she owns, and she does not attempt, e.g., to double-spend her bitcoins or create new funds out of thin air. To this end, Alice's transaction first enters a *pending* state as it gets distributed across the Bitcoin network (Step ②). The exact behavior of the nodes depends on the role they choose to assume [Ano17]: *Full nodes* are nodes that fully validate pending transactions. If a transaction passes all checks, full nodes forward it

to their neighbors, and they keep a copy in a local *memory pool* (*mempool*) [Ano17]. Contrarily, a *light node* strives for processing only a minimal set of transactions considered relevant for the node's operator; hence, the node forfeits the capability to validate transactions independently of other full nodes to save resources [Bit18d]. We will provide more details on Bitcoin's underlying P2P network and its nodes' roles in Section 2.5.

Once the Bitcoin network becomes aware of Alice's pending transaction, it can work toward including the transaction into the blockchain to be accepted by the nodes. The corresponding process of extending the blockchain is called *mining* (Step ③), which is executed by dedicated nodes of the Bitcoin network, the *miners*. Appending a new block to the blockchain requires the miner to invest their resources to solve a computationally hard puzzle. This puzzle requires the miner to create a *Proof of Work* (*PoW*) [Nak08], which entails repeatedly varying the block until its hash value falls below a certain threshold. The threshold is determined by the network's current *mining difficulty*. Full nodes periodically update the agreed-upon mining difficulty based on the rate at which blocks were mined recently; Bitcoin targets an average inter-block interval of ten minutes [BMC⁺15]. The miners continually compete with each other to solve this computationally laborious and unpredictable task, with the prospect that the successful miner is allowed to collect a reward in the form of newly minted bitcoins [Nak08]. This *block reward* incentivizes miners to participate in the mining process. Transaction creators can further attach a *transaction fee* to their transactions, which creates an additional incentive for miners, as they are allowed to collect all fees attached to the transactions they choose to include in their successfully mined block as well. We will discuss the mining process and Bitcoin's reward system in greater detail in Section 2.4.1.

Whenever a miner finds a block, they distribute it across the network (Step ④). Similarly to processing transactions, the full nodes validate the block's correctness by checking, for instance, that the block does not contain double-spending transactions, that it does not attempt to win a too-large block reward, and that their solution to the PoW puzzle matches or exceeds the current mining difficulty. Occasionally, two miners might find valid blocks almost simultaneously. In this case, some full nodes and miners may accept one of these blocks, while others accept the other. This situation is called a *fork* of the blockchain, and both blocks coexist temporarily. The nodes resolve these forks by always considering the *longest chain* valid. As soon as only one of the competing branches gets expanded by a miner, all nodes consider that branch valid. The nodes that previously accepted the now-shorter branch will revert the transactions contained in that branch and then switch to the winning branch, thereby resolving the fork.

Ultimately, Bob receives the new block containing Alice's transaction, sending him her bitcoins, and he can then accept the payment (Step ⑤). However, Bob is advised not to accept Alice's payment right away, because the block was not yet confirmed by subsequent blocks and it may be subject to a fork. The typical recommendation for Bob is to wait until at least six more blocks are mined on top of the block containing the payment [TS16], which corresponds to roughly one hour in wall-clock time.

This example provides a high-level illustration of how Bitcoin operates. However, this simple model is applicable to most blockchain-based systems, since those mostly build upon the same principles and only vary individual aspects of Bitcoin’s initial design. In the remainder of this section, we first distill characteristic properties of (permissionless) blockchains from this high-level overview, before briefly relating Bitcoin’s design to other blockchain models.

Characteristic Properties of Blockchains

Bitcoin pioneered the field of blockchain technology [YMRS18] and blockchains are now widely applied in diverse contexts ranging from insurance management over industrial applications to education. However, Bitcoin is still a prime example to illustrate how blockchains operate at their core due to its comparably simple design. We now point out characteristic properties of permissionless blockchains as a foundation to briefly discuss designs deviating from Bitcoin’s initial design thereafter.

Even though standardization efforts of blockchain technology are still ongoing and did not yield a definitive characterization of blockchains as of now [Wor20], the defining properties of blockchains and the advantages they provide have already been analyzed by multiple technical studies (e.g., [BBSU12, BMC⁺15, TS16, ZXD⁺18, AW19, KAKC20]), reports that primarily target economic stakeholders (e.g., [Gan18, WG18, LHAG19, YSJA21]), or studies from or for governmental agencies (e.g., [Bmwi18, YMRS18, Eu19, BSI19]). In the following, we outline how Bitcoin specifically inhibits the defining properties of blockchain technology (cf. Section 1.1.1).

Decentralization means that the blockchain’s state should not depend on trusted third parties, e.g., to prevent a single point of failure [Eu19]. Notably, it is not assumed that each node trusts every other node [Eu19], but only that a majority of nodes behaves honestly [BBSU12]. Bitcoin relies on a permissionless P2P network to achieve this property on a technical level. However, the true level of decentralization actually achieved by Bitcoin has been questioned in the past [MCR⁺20].

Immutability describes the blockchain’s append-only nature: Once a transaction is recorded on the blockchain, it cannot easily be removed again [KAKC20]. This property prevents a malicious user from reverting transactions after spending their coins in an attempt to regain those coins and trick the other user into accepting a fake payment. Bitcoin makes its blockchain virtually immutable by combining the cryptographic linking of blocks with the computationally challenging PoW puzzle. We note that Bitcoin’s blockchain is only virtually immutable, since modifying data from older blocks only becomes increasingly hard but not impossible. Other publications, thus, refer to Bitcoin as being tamper-evident or tamper-resistant [YMRS18, Eu19], respectively. In this dissertation, we nevertheless use the term “immutability” for simplicity reasons and intend it to mean “virtual immutability” instead.

Public Verifiability means that anybody can validate the correctness of the blockchain’s current state [WG18]. Namely, any interested user must be able to obtain a full copy of all (historic) blockchain data and revalidate past events to arrive at the

current state. As a result, all blockchain data and update rules are publicly accessible [WG18]. In Bitcoin, anybody can operate a full node to obtain and revalidate the full history of its blockchain due to Bitcoin's permissionless approach.

Pseudonymity allows users to hide their identity behind pseudonyms, but different pseudonyms may still be linkable to a common identity [PK01]. Bitcoin manages coin ownership solely via public-private key pairs, which are created locally and in private by the users. Bitcoin users only use their Bitcoin addresses, which are derived from their public keys, as identifiers to transfer bitcoins. As a consequence, blockchains are sometimes said to maintain anonymity [BBSU12, LHAG19, YSJA21], but numerous analyses [MPJ⁺13, RH13, OKH13, SMZ14, FKS15, MN22] have shown that different addresses might be linkable to one user if the underlying blockchain system does not offer additional privacy-enhancing features. Hence, Bitcoin can only provide pseudonymity instead of true anonymity.

Script-based state updates ensure that the system remains flexible to cope with currently unforeseen demands and allow for further innovation [BBSU12], e.g., to foster novel applications. Bitcoin uses a script-based transaction system in which a transaction's outputs express spendability conditions for the associated bitcoins via a stack-based scripting language and a future transaction's input can spend these bitcoins by providing an input script that satisfies the specified conditions.

Other Blockchain Models

Bitcoin's contribution has spawned diverse subclasses of blockchain systems. After having highlighted the characteristic properties of blockchains based on Bitcoin's initial design, we now briefly outline notable variations of blockchain designs.

Permissioned Blockchains. As stated earlier, Bitcoin operates in a permissionless setting where anybody can participate at will. In contrast to this setting, other blockchain systems are *permissioned*, i.e., only selected nodes with verified identities are allowed to maintain the blockchain [WG18]. Relying on a permissioned blockchain is advisable if a consortium of known, but not necessarily mutually trusting, independent stakeholders need to interact with each other [WG18]; such blockchains are also referred to as *consortium blockchains* [YMRS18]. Permissioned blockchain systems do not provide the same openness as permissionless blockchain systems do, but this restriction allows them to operate much more efficiently; for instance, expensive PoW puzzles are only required in permissionless blockchains and the known nodes of a permissioned blockchain can instead use consensus protocols that are less computationally demanding [KAKC20]. Notably, nodes of a permissioned blockchain can be transparent about their activities to still allow for public verifiability [WG18], but this transparency is optional in this setting. Examples of platforms for establishing permissioned blockchains are Hyperledger Fabric³ or Quorum.⁴

³<https://www.hyperledger.org/use/fabric>

⁴<https://consensus.net/Quorum>

Smart Contracts. Bitcoin uses a stack-based scripting language in its transactions to express and satisfy spendability conditions of bitcoins. Other blockchain systems, most notably Ethereum [But13, Woo14] and Hyperledger Fabric, provide richer scripting experiences through *smart contracts*. Initially proposed by Nick Szabo [Sza94], smart contracts are meant to encode the terms of a contract with actions to be executed when the respective conditions are met. Ethereum uses a Turing-complete programming language to create smart contracts and deploy them to the blockchain, i.e., all Ethereum full nodes become aware of all deployed contracts [But13]. Instead of being restricted to transferring funds, transactions can now call functionality of a smart contract, which is then locally executed by all full nodes to update the current state [Woo14].

Privacy-focused Blockchains. As we have discussed earlier in this section, Bitcoin allows for pseudonymous payments but cannot provide full anonymity, as individual payments may be linked to a shared identity that might be deanonymized via side-channel information. Besides the emergence of centralized [BNM⁺14] and decentralized [Max13a, RMSK14, ZMH⁺18] *mixing services*, which allow users to re-anonymize traced bitcoins, also dedicated *privacy-focused blockchain designs* were proposed. These designs cryptographically ensure that payments cannot be linked anymore, usually relying on different forms of zero-knowledge proofs (ZKPs). Examples of such privacy-focused blockchain designs are ZeroCoin [MGGR13], ZeroCash [BSCG⁺14], Monero [Sab13], and Mimblewimble [Poe16].

After this brief discussion of alternative blockchain designs that emerged since Bitcoin’s inception, we now focus on Bitcoin’s technical background again for the remainder of this chapter. This background serves as the foundation for our contributions in the following chapters.

2.2 Blockchain Structure

The blockchain is the central component of Bitcoin’s design, as it enables the independent nodes of the Bitcoin network to reach consensus about which transactions were accepted in the past. We first give a high-level overview of the layout of Bitcoin’s blockchain (Section 2.2.1) before presenting the underlying data structure of its blocks (Section 2.2.2) and Merkle trees, which tie transactions to the blockchain (Section 2.2.3), in detail. For the remainder of this dissertation, we will also use the term “the blockchain” to refer to Bitcoin’s blockchain if not stated otherwise.

2.2.1 Overview

As the name suggests, a *blockchain* is a chain of cryptographically interlinked *blocks*. Namely, each block references its direct predecessor to form the blockchain. Furthermore, each block appends new transactions to the blockchain. Figure 2.2 visualizes the basic layout of the blockchain.

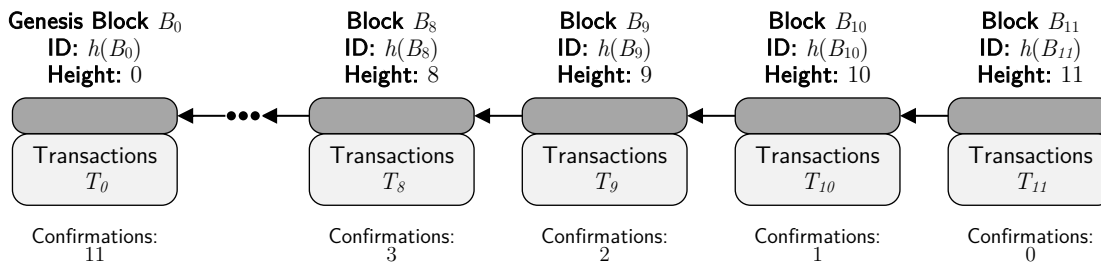


Figure 2.2 The blockchain consists of a list of cryptographically interlinked blocks. Each block adds new transactions to the blockchain and confirms the validity of all its predecessor blocks.

Blocks can be addressed in one of two ways within the blockchain, either using their *block identifier* or their *block height* [Ano17]. The block identifier $ID(B)$ of block B is a cryptographic hash value $ID(B) = h(B)$ derived from the block and is, thus, also referred to as the *block hash* [Ano17]. These block hashes are used to establish *back links* to the blocks' respective predecessors. Because of this cryptographic chaining, any modification to a block *invalidates* its successor's back link. In this case, we say that the blockchain's *integrity* is violated. Contrarily, the block height denotes the block's position within the blockchain. The blockchain starts with a *genesis block* that is hard-coded into any Bitcoin client; the genesis block has a block height of zero. The next block is assigned a block height of one, and so forth. Formally, we say that a block B has a height $HT(B) = i$ if B is valid and points to the genesis block via i back links. For brevity, we also denote the block B with $HT(B) = i$ as B_i . The latest block in the blockchain is also referred to as the *tip*, and we will refer to the latest k -many blocks as the *tail*.

Each block B_i appends a set of transactions $T_i = t_1^i, \dots, t_n^i$ to the blockchain. Whenever a miner finds and disseminates a block, the full nodes and other miners validate that block to decide whether to accept it (cf. Section 2.4.2). The validation process entails checking that the block's back link points to a previously accepted block. Hence, accepting a block serves as a *confirmation* that the referenced block and all its predecessors are valid as well [TS16]. Nodes can thus additionally track the *confirmation count* for each block as an indicator of the likelihood that a particular block is already immutable. For instance, users are advised to wait for at least six confirmations before considering a block's contained transactions final [TS16].

Based on this overview, we can present the blockchain's underlying main data structures in the following. First, we present the structure of individual blocks and how that structure helps achieve immutability (Section 2.2.2). Afterward, we give a brief introduction to Merkle trees [Mer87] and discuss how Bitcoin uses them to cryptographically tie transactions to a block (Section 2.2.3).

2.2.2 Block Structure

In this section, we present the underlying data structure of a Bitcoin block [Bit10a]. The data stored in a block solves four crucial tasks: It (a) appends new transactions to the blockchain, (b) proves that the miner correctly solved the PoW puzzle,

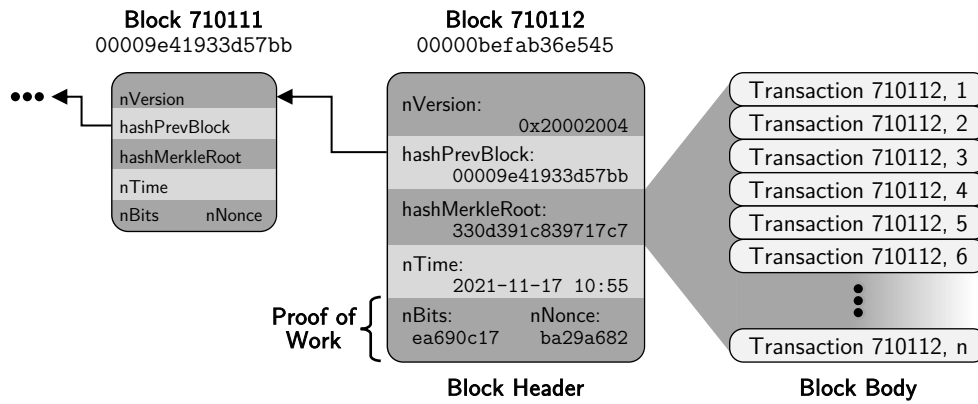


Figure 2.3 Each Bitcoin block consists of a header and a body. The block header contains a version field (`nVersion`), the back link pointing to the block’s predecessor (`hashPrevBlock`), the root of the Merkle tree over the block’s transactions (`hashMerkleRoot`), a timestamp (`nTime`), the current mining difficulty (`nBits`), and a nonce used for the mining process (`nNonce`). The body consists of the list of the block’s transactions, which is cryptographically tied to the block header via the Merkle root.

(c) conveys the block’s approximate (real-world) creation time, and (d) establishes (ultimate) immutability of previous blocks. Figure 2.3 illustrates the structure of a block, and we now discuss how this structure helps solve these four tasks.

Each block consists of a *block header* and a *body*. The block header has a fixed size of 80 Byte and contains all relevant meta information, whereas the body is an ordered list of the block’s transactions $T = t_1, \dots, t_n$. Blocks have a limited size; the effective maximum *block size* has been subject to debate [Bit15c] and varies between Bitcoin and other Bitcoin-like cryptocurrencies [KKSK19]. Bitcoin originally used a maximum block size of 1 MB (= 10^6 Byte), but block size is now measured using a new *block weight* instead [Bit17c]. This change was made when Segregated Witnesses (SegWit) were introduced, which we will discuss briefly in Section 2.3.4.

The *block header* consists of six fields [Bit10a], and we give both their purpose and their names as defined in Bitcoin’s reference implementation⁵ in the following:

Block Version (`nVersion`). Each block has a version telling clients which rules to use for block validation. When the consensus rules get updated over time (cf. Section 2.4), clients can thus easily check that older blocks were valid before according to older rules. As of BIP 34 [And12b], newly mined blocks have version 2. In our example in Figure 2.3, block B_{710112} has a `Version` value of `0x20002004`. The first half-byte `0x2` indicates that this block is a Version-2 block. The remaining bits of the `Version` field are used to signal the miner’s support of proposed consensus upgrades (cf. Section 2.4.3).

Back Link (`hashPrevBlock`). This field contains the back link in form of the 256-bit-long block identifier of the block’s predecessor. The block identifier is derived

⁵The block header is defined in the class `CBlockHeader` in `primitives/block.h`.

by hashing the predecessor's block header twice using the SHA256 cryptographic hash function [Bit10c]. Note that the block's own block identifier and block height are *not* part of the block header.

Merkle Root (`hashMerkleRoot`). This field holds the block's *Merkle root*, i.e., the root of the Merkle tree constructed over the block's transactions. As we detail in Section 2.2.3, modifying the block's transaction set in any way will change the Merkle root and, thereby, the block's identifier. Hence, the Merkle root cryptographically ties the block's transactions to its header. This way, nodes can detach the processing of block headers from validating and applying a block's transactions, e.g., to enable performance improvements when they are only interested in single payments (cf. Section 2.2.3) or during the initial download of the full blockchain (cf. Section 2.5.4).

Timestamp (`nTime`). Each block has a 4 Byte-long timestamp. Even though Bitcoin is continually adjusted so that blocks are likely found every ten minutes (cf. Section 2.4.1), blocks can be found faster or slower at times. The timestamp provides an additional means to reinforce the correlation between block creation and real-world time. However, this timing is only approximate. Full nodes will tolerate variations of roughly two hours to account for issues related to time synchronization within the Bitcoin network [Bit11b].

Target (`nBits`). The target value, internally simply named `Bits`, defines the difficulty of the PoW puzzle used for finding new blocks [Bit10e, Son19]. More specifically, the target defines the inputs for computing a threshold for the block identifier, i.e., only blocks with a hash value below the current target are considered valid (cf. Section 2.4.1). The target is updated every 2016 blocks based on the expected and actual time it took the Bitcoin network to find the last 2016 blocks [Ano17].

Mining Nonce (`nNonce`). This field contains a number used once (nonce), which is used by the miner during the mining process (cf. Section 2.4.1). The miner can vary this 4 Byte-long nonce in the block header to change the block hash until it falls below the threshold defined by the current target.

This information is sufficient to solve the four crucial tasks of blocks we outlined at the beginning of this section, as we further elaborate in the following:

Each block appends a unique set of transactions $T = t_1, \dots, t_n$ to the blockchain by including the fixed-sized Merkle root constructed from T in the block header. Further, any node receiving the block can check that the miner successfully solved the PoW puzzle by recalculating the block hash from the block header and verifying that the block hash is smaller than the mining target. Since each block also contains a timestamp, which has to lie within reasonable bounds, the blocks have an agreed-upon approximate time when they were mined. Finally, the cryptographic chaining of blocks via the back links allows nodes to quickly detect any tampering with individual blocks. Any modification of the block's header would also change its block hash and thereby invalidate the block's PoW. Furthermore, as we discuss in more

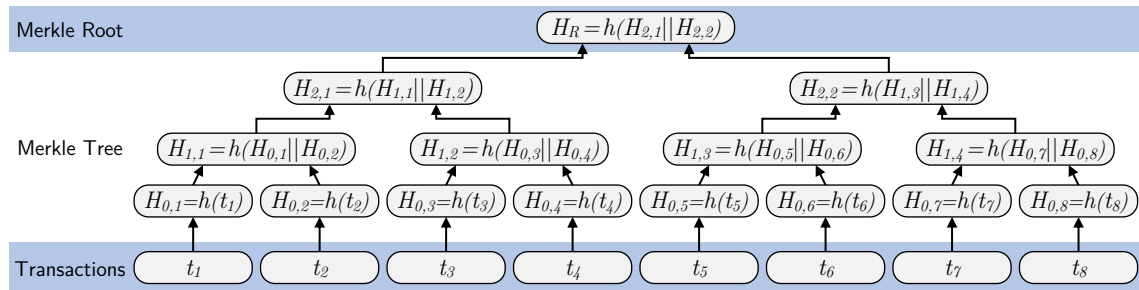


Figure 2.4 Merkle trees [Mer87] are binary hash trees that create a fixed-sized cryptographic “fingerprint” of a list of data items by first computing the hash values of all data items and then recursively hashing two hash values until the single Merkle root remains. In Bitcoin, the Merkle root computed over the transactions of a block’s body is included in the block header to cryptographically tie the body to the header.

detail in Section 2.2.3, changing the block’s transactions in any way also invalidates the PoW by changing the Merkle root. In consequence, whenever an adversary attempts to modify a block, e.g., to reclaim previously spent bitcoins, they have to solve the block’s PoW puzzle again. However, they also have to solve the PoW puzzle again for *all* following blocks as well. Even if the adversary can successfully modify the block, the corresponding block hash will likely be different due to the second pre-image resistance of SHA256. The modification breaks the integrity of the blockchain, as the modified block’s successor will still reference the old block hash in its back link. Thus, the adversary has to update the successor as well, and these changes ripple along the blockchain up until the tip. As long as the adversary does not control a majority of the mining power of the Bitcoin network (51 % attack), they are highly unlikely to catch up with the collective mining efforts of the honest nodes (cf. Section 2.4.3.2). Hence, the combination of PoW and cryptographic back links causes older blocks and the transactions within to become ultimately immutable.

2.2.3 Merkle Trees

We have discussed in the previous section that Bitcoin blocks rely on Merkle trees [Mer87] to tie transactions to the block header using a fixed-sized cryptographic fingerprint of the transaction set. In this section, we first detail how Merkle trees operate in general before presenting two uses they have in the context of Bitcoin.

Merkle trees are binary trees of cryptographic hash values derived from a list of data items, where the root of the Merkle tree, the *Merkle root*, constitutes a cryptographic “fingerprint” over the whole list. In the case of Bitcoin, the data items are the individual transactions $T = t_1, \dots, t_n$ of a block and the cryptographic fingerprint is the Merkle root included in the block’s header.

Figure 2.4 illustrates an example of constructing a Merkle tree [Bit10c] over eight transactions t_1, \dots, t_8 . First, the node hashes the individual transactions and the resulting hash values $H_{0,i} = h(t_i)$ constitute the inputs for constructing the Merkle tree. Bitcoin uses its HASH256 hash function to compute $h(\cdot)$, i.e., the node applies SHA256 twice [Bit10c, Bit10d]. Afterward, the node recursively aggregates the hash

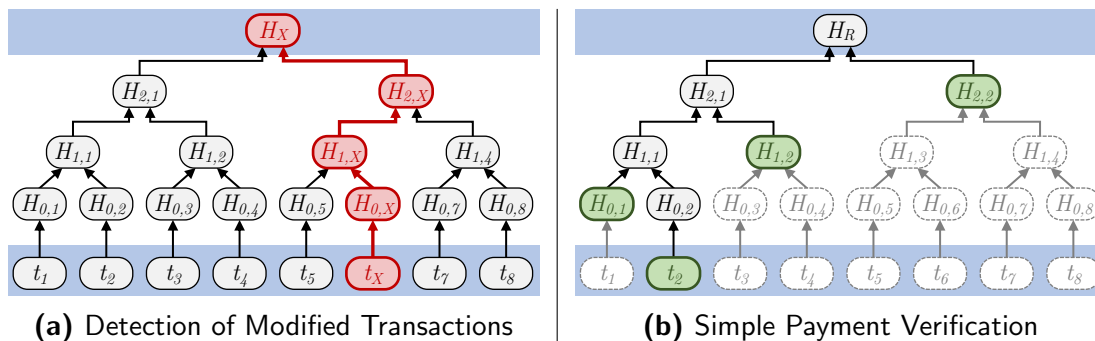


Figure 2.5 Creating a Merkle tree over the transactions contained in a block allows (a) easily detecting any change of a transaction (here: $t_6 \rightarrow t_X$), because the modification propagates up to the tree's root ($M_R \rightarrow M_X$), and (b) quickly validating that a transaction (here: t_2) is part of the block using simple payment verification (SPV), which relies on transferring the relevant intermediate hash values of the Merkle tree instead of the full transaction set.

values from the previous round by concatenating two adjacent hash values and then computing the hash value of that concatenation: $H_{l,i} = h(H_{l-1,2i-1} || H_{l-1,2i})$. If one round has an odd number of input hash values, then the last hash value is duplicated to proceed, i.e., $H_{l,i_{\max}} = h(H_{l-1,i_{\max}} || H_{l-1,i_{\max}})$ in that case. Once only one hash value remains, this value is the Merkle root H_R and the construction concludes.

This construction benefits Bitcoin's design in two ways, as Figure 2.5 illustrates: First, nodes can easily *detect any manipulations* of the transaction set attached to a block. Second, nodes can be *asserted that a specific transaction is included* in the given block without having to know about all of the block's transactions. We explain both of these benefits in more detail in the following.

Detection of Modifications. The main usage of Merkle trees in Bitcoin is their property that modifications of any data item (or transaction) propagate up until the Merkle root. Figure 2.5a visualizes this property. Assume that an adversary modifies transaction t_6 to t_X , e.g., to be able to double-spend their bitcoins by eradicating the fact that they transferred those bitcoins to someone else in that transaction. Building the Merkle tree ultimately relies on the SHA256 hash function (in form of the HASH256 operation). As implied by the second pre-image resistance of SHA256, the adversary cannot modify the transaction while keeping the same hash value, i.e., the Merkle tree will be constructed using $H_{0,X} = h(t_X) \neq h(t_6)$. Similarly, the hash values on higher levels that indirectly depend on t_6 change as well, including the Merkle root. Hence, the Merkle root will change whenever a transaction of a block is modified. Similarly, removing or adding transactions at any position will almost certainly change intermediate hash values of the Merkle tree and, ultimately, change the Merkle root as well. Other nodes can easily detect that the transaction set was modified by recomputing the Merkle tree from the manipulated transaction set and observing a different Merkle root. This error detection would also work when the block header referenced a hypothetical $H'_R = h(t_1 || \dots || t_8)$, i.e., hashing the concatenation of all transactions without any hierarchical composition. However, relying on Merkle trees instead brings the additional advantage that nodes can very efficiently verify single payments without having to know about all transactions of the block, as we discuss in the following.

Simple Payment Verification (SPV). The hierarchical composition of Merkle trees enables a simplified check for the inclusion of particular transactions in a given block, which the Bitcoin white paper called *simple payment verification (SPV)* [Nak08]. Usually, nodes would download and verify the full blockchain and continually monitor new blocks to know about all accepted transactions (cf. Section 2.4.2). However, this overhead is not always desirable. For instance, when Bob wants to quickly verify that Alice really paid for some goods she ordered from him, then Bob is only interested in checking whether or not he received Alice’s payment. As such, it suffices that Bob can verify that the network accepted a payment, sending him the right number of bitcoins. SPV allows Bob to run a light node instead of a full Bitcoin node: He only needs to download and validate the block headers instead of the full blocks (cf. Section 2.5.4) and he can then ask full nodes whether any transactions sent bitcoins to one of his addresses [Ano17]. Figure 2.5b visualizes how Bob can then use SPV to validate that a transaction, e.g., t_2 in our example, is indeed included in a specific block. Instead of obtaining the full transaction set and constructing the Merkle tree, Bob can recompute the correct Merkle root knowing only t_2 and some intermediate hash values from the Merkle tree, which he requests from a full node alongside his transaction. By receiving one auxiliary hash value per level from the full node, i.e., $H_{0,1}$, $H_{1,2}$, and $H_{2,2}$ in our example, Bob can reconstruct the Merkle tree and is assured that t_2 was included in the block he checked for.

2.3 Transactions

Transactions describe the atomic state updates of the blockchain. In this section, we present Bitcoin’s transaction management in detail. After a high-level introduction (Section 2.3.1), we first discuss how transactions are used to keep track of who owns which coins (Section 2.3.2). Then, we give an overview of Bitcoin’s stack-based scripting language SCRIPT, and we show how Bitcoin implements coin ownership on a technical level (Section 2.3.3). Afterward, we outline the relevant changes introduced by Bitcoin’s shift to supporting Segregated Witnesses [Bit17c] (Section 2.3.4). Finally, we present how Bitcoin nodes keep track of the system’s current state by locally managing a set of unspent transaction outputs (UTXOs) (Section 2.3.5).

2.3.1 Overview

Bitcoin users issue transactions to propose state updates to be accepted by the Bitcoin network. Once a miner appends the transaction to the blockchain, all full nodes will apply the changes, usually the transfer of some bitcoins to one or multiple other users, by locally updating their current state. Figure 2.6 illustrates the layout of a transaction and the relation between transactions.

Each transaction consists of *inputs* and *outputs*, where the inputs unlock bitcoins from previous transactions and outputs define spendability conditions for unlocking the coins again in the future. A transaction can have many inputs and outputs, as

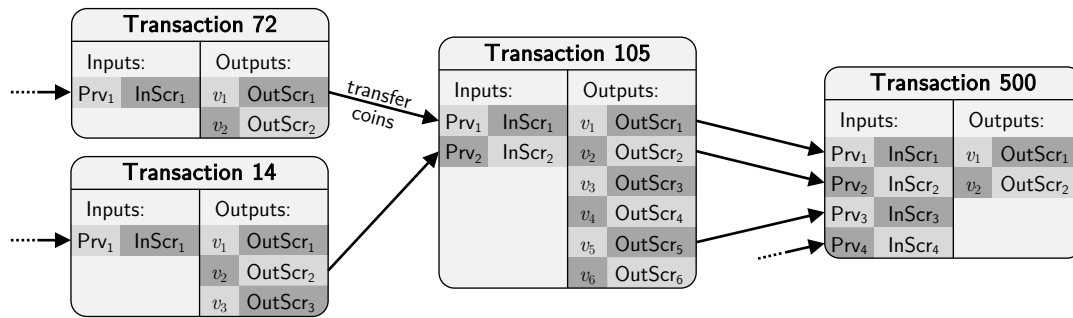


Figure 2.6 Transactions transfer bitcoins between users and consist of (potentially multiple) inputs and outputs. Transaction outputs have an associated value (v), i.e., the number of bitcoins held by the output, and an output script (OutScr) that defines spendability conditions for unlocking the bitcoins. Each transaction input references a previous transaction’s output (Prv) and provides an input script (InScr) that satisfies the corresponding output script. Over time, these references establish the directed and acyclic transaction graph.

long as the total size of the transaction does not exceed 100 kB (pre-SegWit, cf. Section 2.3.4) [SVS18] or 400 kWU (post-SegWit).⁶ Each transaction has a *transaction identifier*, which is derived by computing the transaction’s HASH256 hash value [Ano17]. Bitcoin uses a stack-based scripting language, SCRIPT [Bit10d], to implement the spendability conditions and the checks to satisfy those (cf. Section 2.3.3).

Each *transaction output* consists of a tuple (v, OutScr) , where v denotes the *value*, i.e., the number of bitcoins held by the output, and *OutScr* is an *output script* that defines the condition under which these bitcoins can be accessed by another transaction in the future. As we discuss further in Section 2.3.2, the spendability of bitcoins is usually tied to cryptographic key pairs. Namely, bitcoins are associated with a *Bitcoin address* that is derived from a public key and only the user knowing the corresponding private key is allowed to access these bitcoins. Bitcoin defines a few different *templates* for allowed output scripts, even though SCRIPT allows for composing more complex spendability conditions. We will present these templates in detail in Section 2.3.3.3. Full nodes consider transactions that do not use these templates *non-standard*. They will not relay non-standard transactions to their neighbors, but they will accept when a miner includes them in a block (cf. Section 2.4.2).

Each *transaction input* is a tuple $(\text{Prv}, \text{InScr})$, where *Prv* points to an output of a previous transaction and *InScr* is an *input script* that unlocks that output’s bitcoins by matching the corresponding output script. If the input script matches, we say that the input *spends* the referenced previous output. Each input can only fully spend one output, i.e., *all* bitcoins associated with the output have to be transferred by the transaction. For example, assume Alice wants to send Bob a payment of 10 000 Sat, but she previously only received 5000 Sat in transaction t_a , 3500 Sat in t_b , and 4000 Sat in t_c . Alice has to combine all her funds to pay Bob, i.e., her transaction must include three inputs, each of which unlocks one of the respective output scripts from the transactions t_a , t_b , and t_c . However, these outputs sum up to

⁶The variable MAX_STANDARD_TX_WEIGHT, defined in `policy/policy.h` defines the maximum weight for a pending transaction a full node is willing to forward.

a total of 12 500 Sat, i.e., Alice would overpay Bob by 2500 Sat. To avoid overpaying Bob, Alice includes two outputs in her transaction: in the first output, she sends 10 000 Sat to Bob as intended. In the second output, however, she transfers the remaining 2500 Sat back to herself by generating a new *change address* and sending the excessive funds there.

The repeated transfer of funds between users over time creates a *transaction graph* where new transactions' inputs continually spend outputs from older transactions. In Figure 2.6, for instance, Transaction 500 spends some outputs of Transaction 105, which were previously held by outputs of Transactions 72 and 14, respectively.

In the next section, we go into more detail how Bitcoin implements coin ownership, i.e., the permission to access specific outputs, based on asymmetric cryptography.

2.3.2 Coin Ownership

We now detail how Bitcoin realizes *coin ownership*, i.e., how the network keeps track of who is allowed to spend which bitcoins.

Since Bitcoin is a permissionless system and users can join and leave at will (and without registration), coin ownership cannot be handled via dedicated user accounts. Instead, Bitcoin makes extensive use of asymmetric cryptography to restrict the permission to spend bitcoins to their respective owners. Each user wanting to trade bitcoins has to locally create a public-private key pair first.

Assume that Bob wants to receive some bitcoins from Alice and that he has just created his local key pair. A cryptographic hash value of his public key, the *public-key hash*, will be used to reference Bob's keys on the blockchain. When Alice sends Bob the bitcoins, she includes Bob's public-key hash in her transaction. This way, the Bitcoin network will accept that the transferred bitcoins can only be spent by someone knowing the private key corresponding to that public-key hash. Bob can later on prove his eligibility to spend the coins by creating a digital signature using his private key and including the signature in an input of a future transaction that references Alice's previous output. Bitcoin initially stored the recipient's public key directly in the transaction's output, but shifted to record the public-key hash instead for security reasons (cf. Section 2.3.2.2). In the context of this dissertation, we thus collectively refer to values that are recorded on the blockchain to limit access to coins to specific owners as *on-chain addresses*.

However, Bob will not give Alice his on-chain addresses directly because any typo would prevent him from ever accessing his bitcoins again. This lockout occurs since Bob's private key would not match the on-chain address Alice recorded anymore; we refer to this permanent loss of bitcoins as *burning* them [KKZ20]. Instead, Bob sends Alice his *Bitcoin address*, a human-readable and checksum-protected encoding derived from his public-key hash, which decreases the ambiguity of Bitcoin addresses to humans and allows detecting typos by validating the attached checksum.

In the remainder of this section, we further detail the generation and management of keys (Section 2.3.2.1) and the addressing of coin owners (Section 2.3.2.2).

2.3.2.1 Key Generation and Management

As we discussed before, coin ownership in Bitcoin heavily relies on asymmetric cryptography. Namely, bitcoins are tied to public-key hashes, and spending them requires knowledge of the matching private key. In practice, Bitcoin implements coin ownership via elliptic curve cryptography (ECC) [Mil85, Kob87, BHH⁺14]. In the following, we only give a brief and simplified overview of the relevant aspects of ECC and refer the interested reader to the compendium by Hankerson et al. [HMV04] on which the following overview is based.

Elliptic Curve Cryptography. Cryptographic protocols using elliptic curves offer the same properties as traditional cryptosystems based on linear algebra, such as the Rivest-Shamir-Adleman cryptosystem (RSA) [RSA78], but ECC-based protocols achieve the same security levels with smaller key sizes [LV01, BHH⁺14]. Elliptic curves are defined by equations of the form $E : y^2 = x^3 + ax + b$ over the field \mathbb{F}_p of integers modulo a large prime p . The prime p and the coefficients a, b define a discrete two-dimensional curve that is symmetric around the x -axis. Pairs (x, y) that satisfy E are called *points* on the elliptic curve E . The addition of two points P and Q can be interpreted geometrically: The line intersecting E in the points P and Q will intersect E in a third point R' ; the mirror point R of R' is then defined to be the result of the sum, i.e., $P + Q = R$. Similarly, a special *point doubling* operation enables the repeated summation of a point with itself, which is denoted as $c \cdot P := \sum_{i=1}^c P$; geometrically, the same procedure as above is applied, but the tangent of P is used to determine the mirror point. Alongside $p, a,$ and b , the public parameters of an elliptic curve include a generator G and its order n , i.e., $c \cdot G$ defines distinct points on the curve for $c \in [0, \dots, n - 1]$.

Key Generation and Digital Signatures. Bitcoin users can locally create a new public-private key pair by randomly choosing an integer $d \in [1, \dots, n - 1]$ as the private key and deriving the point $Q = d \cdot G$ as the corresponding public key. The public key Q can now be published, since extracting the coefficient d is assumed to be a computationally hard problem [HMV04]. Bitcoin uses the elliptic curve `secp256k1` [Bro10b], most notably for efficiency reasons [Bit11e]. As a consequence, the private key is a 256 bit-long integer and the public key consists of two 256 bit-long coordinates. Bitcoin makes heavy use of digital signatures to implement coin ownership and relies on the Elliptic Curve Digital Signature Algorithm (ECDSA) [JMV01] for this. ECDSA signatures consist of two elements $(r, s) \in [1, \dots, n - 1] \times [1, \dots, n - 1]$, i.e., r and s are 256 bit-long elements as well.

Serialization of Keys and Signatures. Bitcoin users need to serialize public keys and ECDSA signatures before exchanging them. Bitcoin uses two different serialization methods for keys and signatures when stored on the blockchain: Public keys are serialized using the Standards for Efficient Cryptography (SEC) format [Bro09] and signatures are serialized using the Distinguished Encoding Rules (DER) format [Tel21]. Both formats and their application in Bitcoin are described in detail by Song [Son19] and we only give a brief summary in the following. The SEC format used for public keys allows for an *uncompressed* and a *compressed* serialization; the applied mode is indicated via a single prefix byte [Son19]. In the uncompressed mode, the

public key is serialized as $0x04[x][y]$, where $[x]$ and $[y]$ are 32 Byte-long big-endian integers representing the public key's x -coordinate and y -coordinate, respectively. The compressed mode exploits the symmetry of elliptic curves and, hence, only the x -coordinate is serialized. The prefix byte is then chosen depending on whether the corresponding y -coordinate is even ($0x02$) or odd ($0x03$). As a consequence of the SEC format, all serialized public keys are either 65 Byte (uncompressed) or 33 Byte (compressed) long. For serializing digital signatures, Bitcoin uses the DER format instead. The DER format allows for encoding arbitrary compounds of data values [Mit94] and, therefore, has to maintain more flexibility than the SEC format. A DER-encoded signature starts with a fixed prefix byte as well ($0x30$) but is followed by one byte denoting the length of the encoded value. The encoded value consists of two components representing two integers (r, s). Each component starts with a fixed byte ($0x02$), followed by a variable-length big-endian encoding of the component, i.e., the $0x02$ -marker is followed by a one-byte length field and the encoded component. As a consequence, DER-encoded signatures vary in length, but we can obtain a maximum length for the encoded signature. The components r and s can have at most 256 bit = 32 Byte, but DER prefixes a component with $0x00$ if the first byte would otherwise be larger than $0x80$ [Son19]. Hence, the final encoding of a component is at most 33 Byte long and the resulting signature is between 70 Byte and 72 Byte long. Bitcoin signatures further have a one-byte suffix, called the `SIGHASH` byte, indicating how the input message was chosen when creating the signature [Bit11c, Ano17]. In total, the size of DER-encoded signatures can vary between 71 Byte and 73 Byte in Bitcoin, and since r is randomized, 71 Byte-long signatures can always be achieved [DPNH19].

Wallets. In reality, Bitcoin users do not have to create and manage their keys manually. They rather use a *wallet* software for this purpose [Bit11g]. Strictly speaking, a wallet stores the user's key pairs; however, the wallet software might offer additional services such as unlocking the best-fitting available inputs and handling the change when the user wants to spend a specific amount [Ano17]. Over time, a diverse set of available wallet software for Bitcoin emerged [Bit17d]. For the scope of this dissertation, we only distinguish *hot wallets* [Bit12a] from all other types of wallet software. Hot wallets are all wallets that are accessible via the Internet. These include wallets that are managed by third-party online service providers, which the official Bitcoin Wiki refers to as *custodial wallets*, pointing out that outsourcing the private keys needed to access bitcoins is a bad practice [Bit17d].

We have now outlined how Bitcoin implements coin ownership via public-private key pairs and digital signatures. In the next section, we detail how the public keys are referenced by users and in transactions stored on the blockchain.

2.3.2.2 Addressing Coin Owners in Bitcoin

We have discussed previously in this section that Bitcoin ties coin ownership to private-public key pairs. Namely, transaction outputs lock a number of bitcoins under a public key and the owner can only spend these coins by creating a digital signature using the corresponding private key. Users have to exchange information

about their public keys to be able to create the transaction outputs, transferring funds to the designated recipient. However, errors in this exchange will make the transferred bitcoins inaccessible to the receiver of the payment.

In this section, we outline how Bitcoin represents the owner eligible to spend a transaction output's funds on the blockchain and how Bitcoin addresses harden these representations against potential human errors when exchanging trading information.

On-chain Addresses. As outlined at the beginning of this section, we refer to any (cryptographic) value that is permanently recorded on the blockchain to parametrize the spendability condition of a transaction output as an *on-chain address*. As we discuss further in Section 2.3.3.3, on-chain addresses can refer to cryptographic key pairs (usually owned by a single user) or to hash values of scripts, which serve as an indirection layer to specify more complex spendability conditions.

Historically, Bitcoin transactions referenced a user's public key directly to lock funds in their outputs, i.e., the user only had to use their private key and present the correct signature that can be verified using the corresponding public key. This payment scheme is referred to as *Pay to Public Key (P2PK)*. Bitcoin moved from P2PK to recording only the public-key hash on the blockchain. Bitcoin uses its HASH160 hash function for this purpose [Bit11f], which first applies SHA256 to the public key and then hashes this value again using the RIPEMD-160 hash function [DBP96], i.e., the resulting hash value is only 160 bit (20 Byte) long. This *Pay to Public-Key Hash (P2PKH)* payment scheme then became the standard for most transactions in Bitcoin [Ano17]. This design choice is likely meant to reduce the time frame during which users' public keys are exposed on the blockchain before the corresponding transaction output has been spent successfully [Bit16c]. This exposure time is a threat assuming the presence of sufficiently powerful quantum computers [HVM04], and revealing only the public-key hash before the transaction output should indeed be spent is assumed to mitigate the threat of quantum attacks [ABL⁺18, WEWH22]. When spending a transaction output, the coin owner now has to provide the public key matching the hash value recorded in the output, in addition to the signature proving that they know the correct private key as well. This way, the public key is still published and ultimately recorded on the blockchain, but only once the coin owner intends to spend their coins. As a result, the time frame for an adversary to compute the coin owner's private key from the public key gets considerably shorter and covers only the time from the coin owner publishing their transaction to the Bitcoin network accepting that transaction into the blockchain [Bit16c].

Bitcoin Addresses. After having presented the technical representation of coin ownership in Bitcoin, we now detail how Bitcoin users are involved when trading bitcoins. When Alice wants to make a payment to Bob, she has to know the hash value of Bob's public key in advance to be able to prepare her transaction. However, Alice must be sure to use the correct value, as any mistake on her side causes the bitcoins to be permanently lost. As Bitcoin users generate their key pairs locally (cf. Section 2.3.2.1), the Bitcoin network is unaware of which public-key hashes are valid, and therefore the nodes do not verify Alice's transaction in that regard (cf. Section 2.4.2). To this end, Bitcoin uses *Bitcoin addresses* as a means to encode public keys while accounting for potential human errors during the exchange of on-chain

addresses. The encoding of Bitcoin addresses particularly hardens the underlying public keys against typos or misinterpreted characters.

Bitcoin addresses are derived using an encoding called Base58Check [Bit11a] to achieve these properties. Base58Check operates in two steps to derive the Bitcoin address from a public-key hash (i.e., Bob’s on-chain address in our example): First, the public-key hash is prefixed with a version byte and suffixed with a checksum to obtain a “raw” Bitcoin address of the form [version] [public-key hash] [checksum]. The checksum is computed by applying HASH256 to the public-key hash and taking the first four bytes of the resulting hash value as the 32-bit checksum. This way, Alice will have a chance to detect errors stemming from typos when she revalidates the checksum of the Bitcoin address she receives from Bob. Second, the raw Bitcoin address is encoded using Base58 [NS21] to obtain the final Bitcoin address. Base58 was specifically designed as a purely alphanumeric encoding that additionally avoids character pairs that look alike and thus may provoke typos [NS21]. For instance, the encoding uses neither 0 (digit zero) nor O (uppercase-letter O), and neither I (uppercase-letter I) nor l (lowercase-letter L). Additionally, the encoding avoids non-alphanumeric characters to maximize the usability of Bitcoin addresses (e.g., copying the whole address easily) [NS21].

Some Bitcoin users attempt to further increase the memorability of their Bitcoin addresses by creating *vanity addresses* [Bit12c]. A vanity address is a Bitcoin address with a recognizable part. For instance, `1ComsYSyJF1vbSQrMRTQfnLgMpFapFt9zp` is a Bitcoin address⁷ that starts with a string resembling “COMSYS,” i.e., the affiliation of the author of this dissertation. However, vanity addresses cannot be easily chosen as they are derived from the hash value of a public key. Due to the second pre-image resistance of SHA256 (which impacts HASH256 as well), it is nearly impossible for the address-generating user to obtain the public-private key pair corresponding to a chosen input for the Base58Check encoding. Instead, the user has to repeatedly iterate potential private keys until the resulting Bitcoin address starts with the desired prefix, akin to a (partial) brute-force attack [Ano17].

We have now established the foundation for Bitcoin’s management of coin ownership. In the next section, we detail Bitcoin’s scripting system and present how coin ownership is implemented on a technical level using the scripting language SCRIPT.

2.3.3 Scripting System

After we have established how Bitcoin manages coin ownership on a technical level using public-private key pairs in the last section, we now give an introduction to Bitcoin’s stack-based scripting language, SCRIPT [Bit10d], which is used to implement the spendability conditions in transaction outputs and the eligibility proofs provided in transaction inputs, respectively. We first provide a basic overview of how SCRIPT is embedded in Bitcoin’s transaction management (Section 2.3.3.1) and then give an example of how Bitcoin nodes evaluate attempts to spend coins of a P2PKH transaction output, i.e., coins that are locked by a public-key hash (Section 2.3.3.2).

⁷Do not send bitcoins to this address.

Finally, we give an overview of the reoccurring patterns of scripts that are accepted by Bitcoin nodes as standard transaction types (Section 2.3.3.3).

2.3.3.1 Scripting Language Basics

As we discussed earlier, Bitcoin transactions consist of inputs and outputs, where outputs lock coins behind a spendability condition and inputs unlock some coins of a previous transaction output by providing a proof of eligibility that the output can be spent. To implement these checks, Bitcoin introduced a stack-based scripting language called SCRIPT that operates on binary scripts [Bit10d]. SCRIPT is not Turing-complete and, intentionally, does not feature loops [Bit10d]. The idea behind SCRIPT is that full nodes can locally validate that an input can spend the referenced previous output based on the attached scripts; hence, introducing loops could potentially enable DoS attacks against all full nodes via a single script.

This validation is realized by attaching an *output script* codifying a spendability condition to each transaction output and an *input script* providing the inputs for satisfying a spendability condition to each transaction input. Because the spendability condition is usually tied to a public key and satisfying the condition requires creating a digital signature using the corresponding private key, an output script is also referred to as `scriptPubKey` and an input script is called `scriptSig`.

SCRIPT offers two fundamental operations, *pushing data* onto the stack and *operations* manipulating the stack. Nodes execute a script byte-wise, i.e., they interpret one byte as a so-called *opcode* and act accordingly. An opcode can either instruct the node to push the next bytes onto the stack or to compute a predefined function based on the stack's current state. In this dissertation, we follow the typical convention and denote opcodes, except for certain push operations, as `OP_*` [Bit10d].

A push operation specifies the length of the payload to be pushed onto the stack. For instance, the unnamed hexadecimal opcode `0x06` tells the Bitcoin node to read the next six bytes and push them to the stack as one element. In general, the opcodes `0x01–0x4b` push an element of length 1–75 Byte onto the stack, respectively. Further, there are also dedicated opcodes for pushing larger amounts of data or constant values onto the stack, e.g., `OP_PUSHDATA2` tells the node that the following two bytes contain the length of the following element to be pushed onto the stack and `OP_8` pushes the constant value 8 onto the stack. Even though these opcodes theoretically allow for pushing larger elements onto the stack, a single stack element is effectively limited to a size of 520 Byte [SVS18].

The other operations execute small functions and manipulate the stack. For example, the `OP_HASH160` opcode consumes the top stack element, computes its HASH160 hash value, and pushes the result onto the stack. As we will see in the next section, this opcode is heavily used to validate P2PKH transactions, as public-key hashes are derived using HASH160. A plethora of other, and sometimes less frequently used, opcodes are defined for SCRIPT as well [Bit10d]. For instance, `OP_SWAP` swaps the two topmost stack elements and `OP_ADD` computes their sum. However, multiple initially defined opcodes have been disabled due to concerns of the potential impact of faulty

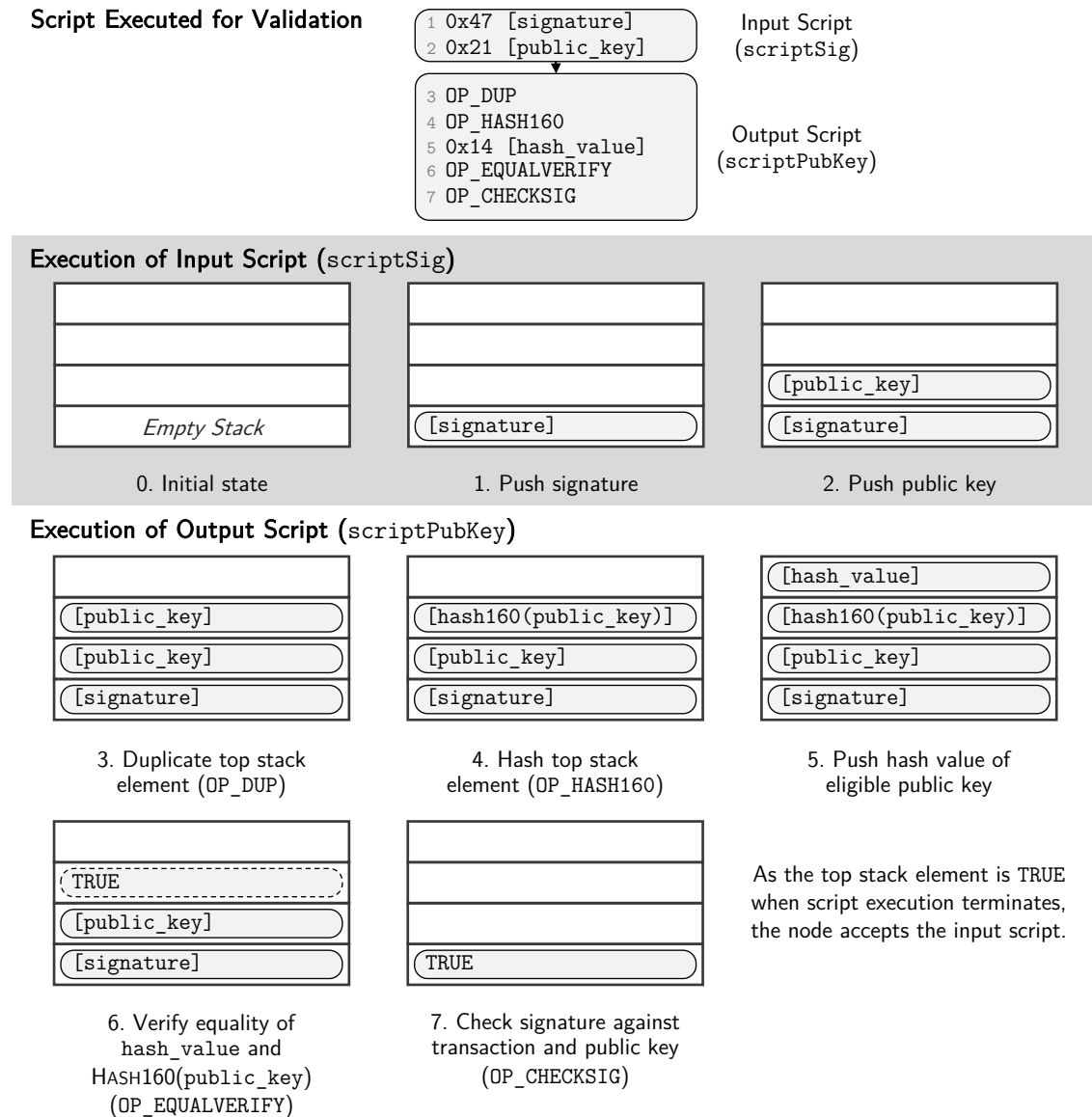


Figure 2.7 Bitcoin nodes validate a transaction input's eligibility to spend a previous transaction's output by concatenating and executing the input script (scriptSig) and output script (scriptPubKey). This figure shows the stack when spending a P2PKH output. An input is only valid if the stack is empty or the top stack element is non-zero (e.g., TRUE, a literal 1) after executing the last line.

implementations, e.g., a previous bug regarding OP_LSHIFT [Bit10d]. Contrarily, some opcodes (OP_NOP1, OP_NOP4–OP_NOP10) are reserved for future use [Bit10d].

An important step of validating pending transactions (cf. Section 2.4.2) is to check the correctness of each of the transaction's input scripts. To validate whether a transaction input correctly spends a previous transaction output, full nodes of the Bitcoin network concatenate the corresponding input script and output script and execute the resulting code. The input passes this check when no opcode explicitly invalidated the execution and the top stack element is non-zero [Bit10d, Ano17].

In addition to their different roles, input and output scripts are also subject to different restrictions. Input scripts must only consist of push operations, and the size per script is limited to 1650 Byte [SVS18]. Contrarily, output scripts are not inherently restricted in that regard, i.e., they can hypothetically use all non-disabled opcodes and their size is only restricted by the maximum transaction size (cf. Section 2.3.1). However, as we discuss further in Section 2.3.3.3, there are accepted *standard patterns* for output scripts. Full nodes shall only relay transactions when all outputs follow these patterns (cf. Section 2.4.2).

In the next section, we first give an example of the verification process of a P2PKH payment before surveying the accepted standard patterns afterward.

2.3.3.2 Example of Script Evaluation: P2PKH

After our overview of SCRIPT in the previous section, we now walk through an example execution that illustrates how Bitcoin nodes validate that a user is allowed to access the coins of a P2PKH output. Figure 2.7 shows both the output script (internally called `scriptPubKey`) using the P2PKH pattern (cf. Section 2.3.3.3) and the corresponding input script (`scriptSig`) that satisfies the output script's spendability condition. Bitcoin nodes independently validate the input script against the output script by concatenating both scripts (`scriptSig||scriptPubKey`) and executing the result. The figure further illustrates the stack after executing each line of the scripts.

The input script first provides the spending user's signature (Line 1) and the public key (Line 2) to match the public-key hash specified in the P2PKH output, and both elements are pushed onto the stack before executing the output script on these inputs. Afterward, the output script first duplicates the top stack element (`OP_DUP` in Line 3), i.e., the public key submitted by the spending user, and then computes its `HASH160` value (`OP_HASH160` in Line 4). This hash value is then checked against the eligible public-key hash by first pushing the reference public-key hash onto the stack (Line 5) and then verifying that both hash values are equal (`OP_EQUALVERIFY` in Line 6). The operation `OP_EQUALVERIFY` combines the operation `OP_EQUAL`, which removes the top two stack elements and pushes `TRUE` (implemented by pushing a literal 1 onto the stack) if both elements are equal and `FALSE` otherwise, and `OP_VERIFY`, which aborts the execution if the top stack element is *not* `TRUE` and removes the checked stack element [Bit10d]. Hence, the operation pushes `TRUE` onto the stack and immediately removes that result within the same operation in our example. At this point, the script ensured that the user provided the right public key. In the final step, the script attempts to verify the provided signature using that public key (`OP_CHECKSIG` in Line 7). The signature is created over parts of the transaction itself [Ano17] to enable the validating nodes to check it autonomously. By default, the user hashes and then signs the whole transaction (except for the input scripts that contain the final signatures), but different approaches are available; the user specifies how they computed the signature in a `SIGHASH` byte appended to the signature [Bit11c, Ano17]. As `OP_CHECKSIG` pushes `TRUE` onto the stack after successfully validating the signature and removing the public key and signature from the stack, the script evaluation succeeds, and the Bitcoin node accepts the input script.

Pattern	Description	Year Added	Specification	Input Script (scriptSig)	Output Script (scriptPubKey)
P2PK	Signature must match public key	2009	Bitcoin Wiki [Bit10d]	[s1] [signature]	[p1] [public_key] OP_CHECKSIG
P2PKH	Signature and public key must match public-key hash	2009	Bitcoin Wiki [Bit10f]	[s1] [signature] [p1] [public_key]	OP_DUP OP_HASH160 0x14 [hash_value] OP_EQUALVERIFY OP_CHECKSIG
P2MS	Signatures for m -out-of- n public keys are required	2011	BIP 11 [And11]	OP_0 [s1] [signature_1] ... [s1] [signature_m]	OP_m [p1] [public_key_1] ... [p1] [public_key_n] OP_n OP_CHECKMULTISIG
P2SH	Input must provide and satisfy the script matching the hash value	2012	BIP 16 [And12a]	[script inputs] [S1] [serial_script]	OP_HASH160 0x14 [hash_value] OP_EQUAL
OP_RETURN	Only add data; cannot be spent	2014	Changelog [Bit14a]	<i>n/a</i>	OP_RETURN [o1] [payload_data]
P2WPKH	SegWit variant of P2PKH	2015	BIP 141 [LLW15]	<i>empty</i>	OP_0 0x14 [witness_hash]
P2WSH	SegWit variant of P2SH	2015	BIP 141 [LLW15]	<i>empty</i>	OP_0 0x20 [witness_hash]
Coinbase	Mint new bitcoins; first transaction in a block	2012	BIP 34 (v2) [And12b]	0x03 [block_height]	<i>any of the above</i>

[s1] is the length of a signature in DER format plus one SIGHASH byte (between 71 Byte and 73 Byte) [DPNH19].
 [p1] is the length of a public key in SEC format, either uncompressed (65 Byte) or compressed (33 Byte).
 [S1] is the length of an encapsulated output script (scriptPubKey) of at most 520 Byte.
 [o1] is a flexible payload length for OP_RETURN outputs of at most 80 Byte.
 OP_m, OP_n refer to the opcodes that push a constant number (m and n) onto the stack.

Table 2.1 List of the standard script patterns as defined by the reference implementation of Bitcoin Core. We additionally note when each pattern was formally introduced.

2.3.3.3 Standard Patterns for Scripts

Bitcoin’s script-based state updates enable flexible spendability conditions. In practice, however, Bitcoin defines a list of *standard* payment schemes and full nodes should reject any transaction not matching these criteria as *non-standard*.⁸ Namely, Bitcoin’s reference implementation has a *standardness check* for pending transactions; this involves, among other checks (cf. Section 2.4.2), that all output scripts must use one of a few accepted opcode sequences. If at least one output script does not match one of these *standard payment patterns*, then the full nodes should reject the transaction. Furthermore, each block has a *coinbase transaction*, which mints new coins and has a special type of input script. As this input script is also standardized, we include coinbase transactions in our overview. Table 2.1 details the opcode sequences of all standard payment patterns and coinbase input scripts and provides the year and sources for the current specification of the respective pattern. In total, we distinguish eight standard patterns:

Pay to Public Key (P2PK). The simplest payment pattern only specifies a public key in the output script. To spend a P2PK output, a user has to provide a valid

⁸The patterns for standard payment schemes are defined in `script/standard.cpp`; a succinct list of standard schemes can be found in the function `GetTxnOutputType()`.

signature created with the corresponding private key in their input script. The earliest mining software for Bitcoin created coinbase transactions with P2PK outputs, and thus already the genesis block uses this payment scheme. Full nodes still accept P2PK outputs, but they are declared obsolete [Bit10d].

Pay to Public-Key Hash (P2PKH). P2PKH is considered the standard payment scheme [Bit10d]. This scheme ties the spendability of bitcoins to a public-key hash instead of a public key. We explained the execution of spending a P2PKH output in detail in Section 2.3.3.2. Together with the signature, a user now has to also specify the public key matching the public-key hash in their P2PKH input. P2PKH has replaced P2PK because basing payment addresses on public-key hashes (Bitcoin addresses) instead of public keys reduces their size and, further, P2PKH only exposes public keys to potential quantum attacks on ECDSA for a shorter time (cf. Section 2.3.2.2) [Bit10d].

Pay to Multi-Signature (P2MS). This payment scheme allows locking coins such that multiple users have to agree to spend them. A P2MS output script specifies n public keys as well as a parameter m to implement an m -out-of- n spendability condition, i.e., the corresponding input script needs to provide valid signatures for m of the given public keys. However, the Bitcoin reference implementation restricts valid P2MS outputs to at most m -out-of-3 payments.⁹ All P2MS input scripts have to start with an `OP_0` opcode; this is due to a bug in the implementation of `OP_CHECKMULTISIG` [Bit21b]. BIP 11 defines the P2MS payment scheme [And11].

Pay to Script Hash (P2SH). The P2SH payment scheme was defined in 2012 in BIP 16 and was meant to simplify the process of sending bitcoins [And12a]. When Alice wants to send Bob some bitcoins without using P2SH, she has to be fully aware of how Bob may intend to access the transferred coins in the future. For instance, if Bob negotiated with Claire and Danny that they want to manage the coins jointly using a four-eyes approach, then Alice has to create a 2-out-of-3 P2MS output on Bob's behalf, whereas a simple P2PKH output would suffice if Bob was the sole receiver of the coins. To account for that, P2SH shifts the responsibility of specifying the spendability condition from the sender to the receiver (i.e., the future owner and manager) of the transferred coins: Bob first creates a script codifying the true spendability condition, also called the *redeem script* [Ano17]. He then derives a *script address* by computing the HASH160 hash value of the redeem script and using the same procedure to encode this hash value as he would use to derive a Bitcoin address from his public-key hash (cf. Section 2.3.2.2). The only difference here is that he uses version `0x05` instead of the version `0x00` used for P2PKH payments [Bit12b]. Alice now only has to extract the hash value of the redeem script from the script address to create her transaction output, i.e., she can remain oblivious of Bob's true policy for accessing his coins. Bob can subsequently spend the coins by disclosing the redeem script matching the hash value and providing the inputs satisfying the redeem script's spendability condition. In addition to

⁹This check is made in the function `IsStandard()` in `policy/policy.cpp`.

simplifying the process for Alice, P2SH also unburdens her financially. As we discuss further in Section 2.4.1, transaction creators pay per-byte transaction fees. If Bob required a complex policy for accessing his coins, Alice would have to create a longer output script without P2SH, resulting in higher fees for her. With P2SH, however, Alice's output script has a constant size regardless of Bob's policy and, instead, he has to pay the additional fees once he publishes the redeem script as part of his next payment.

OP_RETURN. Transaction outputs starting with the `OP_RETURN` opcode, followed by only push operations, allow the transaction creator to attach a small chunk of arbitrary data to their transaction. Any transaction may have at most one `OP_RETURN` output; otherwise, the transaction is invalid. Even though full nodes configure the maximum size for `OP_RETURN` output scripts they are willing to accept, the default setting is 83 Byte,¹⁰ which covers the `OP_RETURN` opcode, two bytes for a push operation, and a payload of at most 80 Byte. The Bitcoin client relays transactions containing an `OP_RETURN` output since version v0.9.0 [Bit14a], and since version v0.12.0 it allows for splitting the payload into fields via multiple smaller push operations [Bit16a].

Pay to Witness Public-Key Hash (P2WPKH). The Segregated Witness (SegWit) update was specified in 2015 in BIP 141 [LLW15] and deployed to the Bitcoin network in 2017 [Bit17c]. SegWit presented the ultimately agreed-upon solution to Bitcoin's controversy about maximum block sizes [Bit15c]. As we discuss further in Section 2.3.4, SegWit decouples the *witnesses* proving eligibility to unlock an output from the transaction itself and ensures that both legacy nodes and SegWit-aware nodes accept inputs accessing SegWit outputs. In this vein, the P2WPKH payment scheme constitutes the SegWit variant of P2PKH: A P2WPKH output only specifies a SegWit version (`OP_0`) and the eligible public-key hash (i.e., a 20 Byte-long `HASH160` hash value). The corresponding input script remains empty. Legacy nodes blindly accept the input due to the non-zero public-key hash being the top stack element. SegWit-aware nodes will additionally inspect the detached witness, a signature and the public key in the case of P2WPKH, during the validation process.

Pay to Witness Script Hash (P2WSH). This payment scheme is the SegWit variant of P2SH [LLW15]. P2WSH operates analogously to P2WPKH; the only differences are that (a) the witness for unlocking a P2WSH consists of the required inputs for the redeem script of the underlying P2SH scheme and the redeem script itself and (b) the hash value recorded in the output script is computed using `HASH256` instead of `HASH160`, i.e., the hash value is 32 Byte long.

Coinbase. Coinbase transactions do not describe a payment scheme, but Bitcoin's way of creating new coins (we discuss the process of creating coins further in Section 2.4.1). On a technical level, the first transaction of each block must be

¹⁰This setting is configured via the variable `nMaxDatacarrierBytes`, which defaults to `MAX_OP_RETURN_RELAY = 83`; the corresponding check is performed in the function `IsStandard()` in `policy/policy.cpp`.

a coinbase transaction. The successful miner may lock the newly created coins with any output script, but the input script has a standardized form. Since 2012, Bitcoin requires the miner to store the block’s height in the coinbase input using a special encoding as proposed by BIP 34 [And12b]. However, as we further elaborate in Section 3.3.1, the coinbase input allows for additionally storing a few bytes of arbitrary data. Namely, a coinbase input has a maximum total length of 100 Byte¹¹ [Son19], and we refer to all arbitrary data following the block height as the *coinbase field*.

Summary. Bitcoin full nodes should only accept these eight script patterns. Overall, we can classify the patterns into (a) output scripts for payment schemes that require unlocking coins while providing cryptographic evidence of coin ownership in the input script (P2PK, P2PKH, P2MS, P2SH), (b) SegWit variants of those schemes (P2WPKH and P2WSH), (c) unspendable `OP_RETURN` outputs augmenting a transaction with arbitrary data, and (d) coinbase inputs. Because of their relative similarity to each other, we also collectively refer to P2PK, P2PKH, P2MS, and P2SH as “*Pay to X*” (*P2X*) in this dissertation.

2.3.4 Segregated Witnesses

Bitcoin has been subjected to prolonged scalability debates regarding the maximum size of admissible blocks [Bit15c, Mor17]. Traditionally, Bitcoin blocks are restricted to a size of 1 MB and several stakeholders proposed different increases to this limit to improve the achievable transaction throughput [Bit15c, Mor17]. The final solution deployed in Bitcoin Core [Bit15c] is called *Segregated Witnesses (SegWit)*. SegWit detaches the *witnesses*, i.e., the cryptographic evidence needed to validate that a payment was made by a rightful coin owner, from the rest of a transaction and thereby does not count into the traditional limit for the block size. At the same time, SegWit maintains a certain degree of backwards compatibility to legacy nodes. In this section, we give a brief overview of SegWit and outline its backward compatibility as well as its impact on the maximum block size.

SegWit is specified over multiple BIPs [Bit17c], but our overview focuses on the main proposal found in BIP 141 [LLW15]. The underlying principle of SegWit is the separation (or “segregation”) of the flow of bitcoins between transactions and the witnesses used to legitimate transfers [LLW15]. SegWit achieves this separation by (a) detaching the witnesses from the input scripts recorded on the blockchain and instead storing them in a new data structure of extended transactions (specified in BIP 144 [LW16]), (b) introducing a new witness tree in addition to the Merkle tree of a block, which cryptographically binds all witnesses to the block, and (c) establishing new payment schemes that tie the spendability of the associated coins to providing the respective witnesses (P2WPKH and P2WSH, cf. Section 2.3.3.3).

One fundamental problem for such proposal is the deployment of changes that affect the consensus rules, i.e., how full nodes accept or reject proposed blocks. SegWit was

¹¹This policy is defined in the function `CheckTransaction()` in `consensus/tx_check.cpp`.

deployed via a *soft fork*, an update mechanism to introduce stricter validation rules while retaining some backwards compatibility by keeping legacy nodes oblivious of the updated rules (cf. Section 2.4.3). For instance, as we discussed in Section 2.3.3.3, spending a P2WPKH output with an empty input script is valid even for legacy nodes. However, legacy nodes would accept *any* attempt to spend the P2WPKH output and, thus, legacy nodes have to rely on a majority of updated nodes to agree on filtering out transactions that are invalid according to the updated rules. The witness tree is influenced by the deployment via a soft fork as well: Embedding the root of the witness tree into the block header (e.g., next to the Merkle root) would cause legacy nodes to reject SegWit blocks. Instead, miners include the root of the witness tree in their coinbase field, which is allowed to include arbitrary data and is ignored by legacy nodes (cf. Section 2.3.3.3).

Through SegWit, legacy nodes are not aware of the witnesses anymore, and thus the nodes also validate the block size without accounting for the witnesses. This way, SegWit can circumvent the hard-coded maximum block size of 1 MB. Instead, BIP 141 defines a new unit, the *weight unit (WU)* [LLW15], where 1 Byte of witness data is counted as 1 WU, but 1 Byte of non-witness data (or witness data prior to the SegWit update) is effectively counted as 4 WU [Bit17c]. BIP 141 then defines a maximum block weight of $4 \cdot 10^6$ WU = 4 m WU. As a consequence, the SegWit update allowed the Bitcoin network to shift to larger blocks without introducing network-breaking changes.

2.3.5 Managing Unspent Transaction Outputs

As of now, we have presented how transactions are persisted immutably on the blockchain. However, not all blockchain data is relevant for validating pending transactions at all times: if all outputs of a transaction have been spent, any future transaction trying to spend one of its outputs must be an attempted double spending. Hence, information recorded on the blockchain becomes obsolete over time as coins continue to be transferred. In this section, we give an overview of the relevant *local state* a full node maintains to be able to validate pending transactions efficiently.

To this end, a full node maintains a lookup table of all *unspent transaction outputs (UTXOs)* called the *UTXO set* [Ano17]. A UTXO is any transaction output that is not provably unspendable (e.g., `OP_RETURN` outputs are excluded from the UTXO set) and that has not yet been accessed by a subsequent transaction input. The full node uses the UTXO set to validate pending transactions and updates the UTXO set whenever it has received a valid block. Namely, the full node checks whether each input of a pending transaction correctly spends a UTXO. If a referenced previous output is not in the UTXO set, then the node has identified an attempted double spending and rejects the pending transaction. Similarly, if the input does not provide the correct witness for the referenced output, the full node also rejects the pending transaction [Son19]. Once the full node receives a valid block, it executes all contained transactions by removing all now-spent transaction outputs from the UTXO set and adding the new transaction outputs.

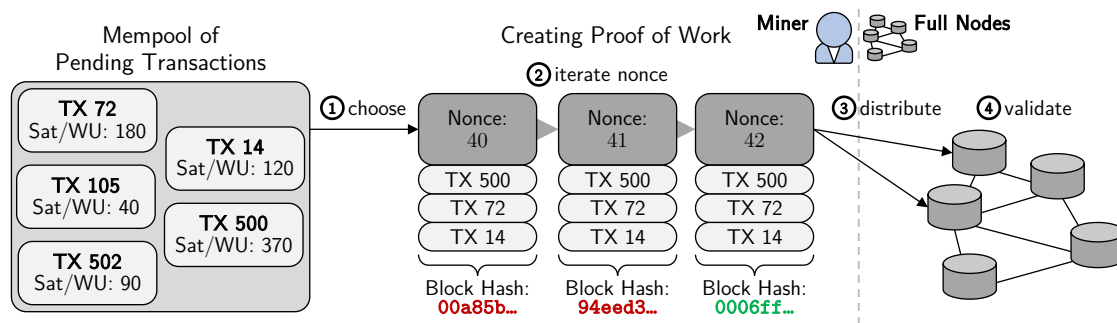


Figure 2.8 Bitcoin’s consensus mechanism comprises four steps. All nodes maintain a local mempool of pending transactions, and the miners choose a subset of pending transactions to include in their block ①. Afterward, the miner creates the Proof of Work (PoW) by iterating the nonce of the block header such that the block hash falls below the threshold defined by the current mining difficulty ②. Once successful, the miner distributes the block to the Bitcoin network ③ and the full nodes accept the block after validating it ④.

Bitcoin Core keeps the UTXO set in a dedicated folder called `chainstate` [Bit21a]. This folder contains a LevelDB [Bit21a] database of all UTXOs and has one row per UTXO [Bit17b]. Bitcoin Core uses a *compression scheme* to reduce the space occupied by UTXOs where possible. This compression scheme builds upon the standard patterns for transaction outputs (cf. Section 2.3.3.3), i.e., instead of storing the whole output script the database only contains its mutable values and the node decompresses the original output script on the fly when validating a transaction spending this specific UTXO. As of now, Bitcoin Core recognizes six cases of compressible scripts, and the first byte of a compressed script denotes the case applied during compression:¹² P2PKH (case `0x00`), P2SH (`0x01`), P2PK with compressed public key (cases `0x02` or `0x03` taken from the public key’s prefix byte, cf. Section 2.3.2.1), and P2PK with an uncompressed public key that is compressed as well to be stored in the UTXO set (`0x04` if the public key’s *y*-coordinate is even or `0x05` if it’s odd).

With this overview of the UTXO set, we conclude our introduction to Bitcoin’s transaction management. In the next section, we present how nodes establish a network-wide consensus about the blocks and transactions to be accepted.

2.4 Consensus

After having detailed Bitcoin’s transaction system in the last section, we now present how the nodes establish consensus about which transactions shall be irrevocably accepted into the blockchain. Figure 2.8 illustrates the different steps of establishing consensus in Bitcoin: First, miners have to ① choose a subset of currently pending transactions for inclusion in their next block. Then, the miner ② iterates the nonce, which is part of the block’s header, until the block is eligible and the miner has found a Proof of Work (PoW) for their block. The Bitcoin network agrees on a current mining difficulty, and the block hash of the current candidate block must fall below

¹²Output scripts are compressed in the function `CompressScript()` in `compressor.cpp`.

a threshold defined by this difficulty. Once the miner found a nonce that solves this PoW puzzle, they can ③ distribute it to the other nodes in the Bitcoin network and all full nodes ④ validate the block's correctness before accepting it into their local blockchain copy.

In this section, we present the consensus establishment by first giving an introduction to the mining process and the miners' incentives to participate in this process (Section 2.4.1). Then, we give a brief overview of the full nodes' validation process for pending transactions and newly announced blocks (Section 2.4.2). Finally, we outline how Bitcoin resolves accidental blockchain forks, a side effect occurring when miners find blocks roughly at the same time, and we discuss intentional forks triggered by updates to the initially agreed-upon validation rules (Section 2.4.3).

2.4.1 Mining Process

Bitcoin's mining process ensures that transactions become virtually immutable over time by requiring nodes to solve a computationally hard puzzle in order to append a new block to the blockchain. Requiring this *Proof of Work (PoW)* and the cryptographic linking between blocks ensures that older blocks effectively cannot be replaced by an adversary controlling only a minority of the computation power dedicated to the mining process (cf. Section 2.2.2). In this section, we give a brief overview of the incentives to become a miner, i.e., dedicating resources to find new blocks, as well as the process itself, i.e., how miners prepare their blocks and how they solve the PoW puzzle (Steps ① and ② in Figure 2.8).

Incentives. Bitcoin offers two monetary incentives to become a miner, *block rewards* and *transaction fees*. Each successful miner gets a *block reward*, which consists of newly minted bitcoins that the successful miner can transfer to a Bitcoin address under their control via the block's coinbase transaction. The number of bitcoins the miner is allowed to mint is governed by the consensus rules and is halved every 210 000 blocks¹³ (corresponding to roughly four years) [Ano17]. Starting off with 50 BTC per block, miners are rewarded 6.25 BTC per block since block 630 000 (mined on May 11, 2020). Ultimately, block rewards will be so low that effectively no new bitcoins enter circulation under the current rules.¹⁴ Bitcoin offers *transaction fees* as a secondary incentive for miners. If a transaction creator overpays, i.e., not all bitcoins unlocked by the inputs are consumed by the transaction's outputs, the miner adding the transaction to the blockchain is allowed to add the excessive coins to their block reward. As a consequence, miners are incentivized to include transactions yielding larger fees in their block whenever the size of pending transactions in the mempool exceed the maximum block size. Transaction fees are expressed per byte (or WU). The wallet software used to create transactions will estimate feasible per-byte fees [Ano17]. Additionally, third-party services track recent fee rates and

¹³The current block reward is calculated in the function `GetBlockSubsidy()` in `validation.cpp`; the halving interval is defined as an attribute `consensus.nSubsidyHalvingInterval` of the class `CMainParams` in `chainparams.cpp`.

¹⁴Bitcoin Core effectively cuts off block rewards after 64 halvings in `GetBlockSubsidy()`.

pending transactions to make recommendations for slower or faster estimated delays until a transaction will likely be picked up by a miner [Pri20].

Block Preparation. The miner initiates the mining process by preparing a block they wish to include in the blockchain. Most notably, the miner chooses currently pending transactions they are aware of from their local mempool (Step ① in Figure 2.8). As discussed before, the miner is incentivized to select the transactions paying the highest transaction fees, as they further improve the miner’s achievable block reward. Once the transactions and the total block reward are fixed, the miner can prepare a *candidate block* for subsequently solving the PoW puzzle [Ano17]. Namely, the miner has to prepare the block header, i.e., (a) fix the block’s version, (b) insert the back link to the block’s predecessor, (c) prepare the coinbase transaction and then construct the Merkle tree, (d) fill in the current difficulty, and (e) insert the time stamp (cf. Section 2.2.2). Based on this candidate block, the miner can start working on the PoW puzzle. However, the miner has to periodically update the candidate block and start the process anew due to the included time stamp or whenever they become aware of a new block from another successful miner.

Proof of Work (PoW). The mining process mandates that a miner has to solve a PoW puzzle for their block to become valid. The PoW puzzle is based on the proposed block’s block hash (the `HASH256` value of the block’s header, cf. Section 2.2.1) and the network’s current *mining difficulty*. Namely, the mining difficulty specifies a threshold and only blocks with a block hash below this threshold are considered valid regarding the mining difficulty. To attempt solving the PoW puzzle, the miner slightly varies the candidate block without changing its semantics. Here, the miner has two main options. The first and canonical option is to iteratively increment the `nNonce` value in the block header. However, with only four bytes of space in this field (i.e., 2^{32} possible values), the miner might not be able to solve the PoW puzzle using only the `nNonce` field [Ano17]. As a second option, the miner can hence vary additional bytes in the coinbase field (cf. Section 2.3.3.3) to increase the space for varying bits and solve the PoW puzzle [Ano17]. The mining difficulty is updated periodically to readjust the effective inter-block interval [Ano17]. Namely, the whole Bitcoin network should statistically find a new block every ten minutes¹⁵ [BMC⁺15]. Fluctuations in the network’s mining capabilities, e.g., changing numbers of miners or improved mining hardware, could affect the inter-block interval experienced in reality [Ano17], hence making this periodical readjustment necessary. Effectively, the mining difficulty is updated every 2016 blocks (i.e., roughly every two weeks¹⁶) [Ano17]. All Bitcoin nodes are aware of the current (and historical) mining difficulty via the target value (`nBits`) in the block header (cf. Section 2.2.2), which is a space-efficient encoding of the threshold for valid block hashes, i.e., the mining difficulty [Son19].

¹⁵The target inter-block interval is defined as an attribute `consensus.nPowTargetSpacing` of the class `CMainParams` in `chainparams.cpp`.

¹⁶The interval for updating the difficulty has a target value of two weeks, defined via the attribute `consensus.nPowTargetTimespan` of the class `CMainParams` in `chainparams.cpp`. The function `DifficultyAdjustmentInterval()` in `consensus/params.h` then returns the 2016 blocks as a result of dividing `consensus.nPowTargetTimespan` by `consensus.nPowTargetSpacing`.

Once the miner has found a block, it can ③ distribute the block to the other nodes, who can efficiently verify (as part of the ④ validation process) that the block successfully solves the PoW puzzle by computing its block hash and checking that hash value against the current mining difficulty. In the following, we go into further detail about the distribution and validation process of the Bitcoin network.

2.4.2 Distribution and Validation Process

Bitcoin establishes consensus by defining rules that every node uses to individually validate an obtained copy of the blockchain, as well as new incoming blocks and transactions, locally. These rules range from simple format checks over sanity checks, e.g., whether a proposed transaction is in conflict with a previously accepted transaction, to block-level checks, e.g., whether the current mining difficulty (cf. Section 2.4.1) has been adhered to. In this section, we briefly present general principles for validation in Bitcoin, the most relevant validation rules, and consequences on establishing the consensus.

By default, Bitcoin nodes validate incoming information extensively. These nodes are hence referred to as *full nodes* [Ano17]. Full nodes maintain a full copy of the blockchain and perform all validation steps to ensure that they only accept correct information [Ano17]. Contrarily, *light nodes* [Bit18d] only download the block headers and only validate a subset of transactions deemed relevant to them using SPV (cf. Section 2.2.3). As such, light nodes do not contribute to the overall operation of the Bitcoin network, and we hence focus on full nodes in the following.

We now give a brief overview of how the full nodes validate blocks and transactions they receive, as this information is the most relevant for achieving consensus about the current state of the blockchain. We further highlight a relevant asymmetry when full nodes validate pending transactions in contrast to transactions included in a mined block. A full overview of the validation process is maintained in the Bitcoin Wiki [Bit11d].

Block-Level Validation. When a full node receives a newly mined block, the node validates that block before accepting it and relaying it to other nodes. The node performs several checks on the block header and the block as a whole.¹⁷ Basic checks ensure that the block is syntactically correct, has a correct Merkle root, and has a sensible back link [Bit11d], i.e., the block is a valid extension to the blockchain that keeps its structure intact. Additionally, the full node also ensures other, more technical, properties of the newly mined block. For example, the node ensures that the block's time stamp is at most two hours in the future and that the block does not exceed the size limits [Bit11d]. Afterward, the node validates the block's transactions, as we present in the following.

¹⁷The block-level checks are defined in various functions of `validation.cpp`; most notable among those functions are `AcceptBlock()` and `AcceptBlockHeader()`, which call other checks as part of checking whether a block (header) should be accepted or rejected by the node.

Basic Transaction-Level Validation. As part of the validation of a newly mined block, a full node also validates all the transactions contained within. The validation ensures that the first transaction (and only the first) is a correct coinbase transaction, and then proceeds to validate all “common” transactions (i.e., non-coinbase).¹⁸ To this end, the full node has to perform three essential checks whenever validating a common transaction. Namely, the full node has to check that (a) all transaction inputs have not yet been spent, (b) all input scripts can successfully unlock the referenced previous outputs, and (c) the transaction outputs do not spend more funds than unlocked by the inputs [Son19]. Together, these checks ensure that no funds are spent incorrectly. Full nodes use their local UTXO set (cf. Section 2.3.5) to validate the first two rules, i.e., each input has to access coins currently considered unspent by the full node and the input scripts must satisfy the spendability conditions defined by the referenced output scripts (cf. Section 2.3.3). Regarding the last rule, the sum of spent funds must not exceed the unlocked funds. However, the transaction creator can choose to unlock more funds than to be spent by the outputs. Any excessive funds are considered voluntary fees to be collected by the miner (cf. Section 2.4.1). Additionally, the node performs a few technical validation steps, e.g., the transaction must have at least one input and one output [Bit11d]. This concludes the validation of transactions mined into a block. As we discuss in the following, pending transactions, i.e., transactions that have been proposed but were *not* yet included in a block, are examined with considerably more scrutiny.

Validation of Pending Transactions. In addition to thoroughly validating the information they accept into their local blockchain copy, full nodes also constitute the backbone of the Bitcoin network (cf. Section 2.5.1). As such, full nodes already disseminate transactions proposed by users that are still pending, i.e., not included in a block yet. The full nodes conduct additional validation steps to those mentioned before to decide whether to relay or discard a pending transaction¹⁹ [Bit11d]; hence, the validation of transactions to be forwarded tends to be much stricter than the validation of transactions included in a block [BMC⁺15]. For instance, the node rejects pending transactions that are incompatible not only with previously accepted blocks, but also with their local mempool [Bit11d]. Furthermore, full nodes reject transactions with outputs only paying uneconomic “dust amounts” [DPNH19].²⁰ Most notably, however, the node rejects any pending transaction with output scripts deviating from the accepted standard patterns we presented in Section 2.3.3.3 (additionally, coinbase transactions are always rejected as they may only occur in a block, not as pending transactions) [Bit11d]. Since this check is omitted for transactions that are included in a block, non-standard transactions can be accepted into the blockchain despite the full nodes’ rule to reject such transactions. For instance, proposing a non-standard transaction to a miner directly can circumvent this stricter validation step [BMC⁺15]. The output scripts are also not validated

¹⁸These basic transaction-level checks are performed in the function `CheckTransaction()` in `consensus/tx_check.cpp`; the check whether or not a transaction is a coinbase transaction is defined in the method `CTransaction::IsCoinBase()` in `primitives/transaction.h`.

¹⁹These additional checks are performed in the method `MemPoolAccept::PreChecks()` defined in `validation.cpp`.

²⁰What amount accounts for dust depends on the length of the output script and is computed in `GetDustThreshold()` in `policy/policy.cpp`. For P2PKH outputs, the dust threshold is 546 Sat.

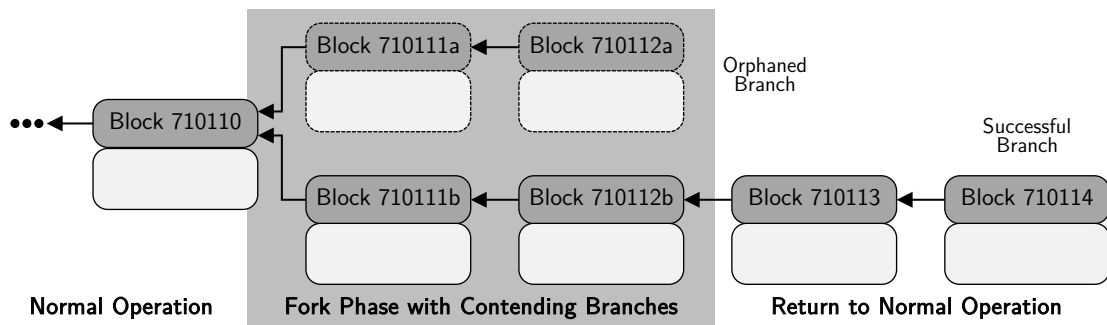


Figure 2.9 Whenever two miners find a new block simultaneously, they create a temporary fork, as some full nodes accept one candidate and other full nodes accept the other candidate as the next valid extension of the blockchain. Also, miners will start mining on both branches, potentially extending the fork further. However, one branch of the fork is expected to outgrow the other (i.e., accumulate more PoW), which causes the losing branch to be abandoned as all nodes perform a rollback and switch to the winning branch to return to normal operation.

beyond this standardness check. Most importantly, the nodes do not ensure that a designated recipient really exists. Since users create their identities locally (cf. Section 2.3.2), full nodes have no means to verify that the on-chain address referenced in a transaction output is correct. Instead, the full node will accept any on-chain address of acceptable length in good faith that some user knows the corresponding private key. Otherwise, the associated bitcoins would be burned, i.e., become inaccessible. For this reason, the encoding of Bitcoin addresses includes a checksum as well to prevent simple typos from causing the permanent loss of currency.

2.4.3 Forks

The probabilistic nature of Bitcoin’s consensus mechanism, which involves the miners competing over finding the next block, inherently bears the risk of collisions. Whenever two miners are successful roughly at the same time, two candidate blocks exist that can be appended to the blockchain. Generally, a *fork* of the blockchain is created whenever the full nodes do not unanimously choose one of the available candidates, i.e., different full nodes accept one candidate or the other simultaneously.

In this section, we give a brief overview of two main types of blockchain forks that can occur. First, we discuss how *accidental forks* can emerge in Bitcoin and how they are resolved (Section 2.4.3.1). Afterward, we briefly consider *intentional forks*, which emerge when full nodes start to disagree about the applicable consensus rules (Section 2.4.3.2).

2.4.3.1 Accidental Forks

Bitcoin’s consensus-finding mechanism relies on the redundant verification of all published information by the full nodes, but also on the perpetual competition between the individual miners to find the next valid block. As solving the PoW puzzle to find the next block is a probabilistic challenge, multiple miners can succeed almost

simultaneously and broadcast their blocks to the Bitcoin network [TS16]. In such situations, the Bitcoin network created an *accidental fork*. Figure 2.9 illustrates the life cycle of such an accidental fork.

As miners operate in isolation,²¹ resulting blocks may be in conflict with each other. Most notably, transactions from both candidate blocks may attempt to access the same coins; this can happen, for instance, if multiple miners include the same pending transaction in their blocks. At the very least, each miner will attempt to send the mining reward to an on-chain address they control [DW13]. In our example, two miners concurrently published the Blocks 710 111a and 710 111b. As miners start mining on top of new blocks as soon as they get aware of them (cf. Section 2.4.1), the fork might be extended over multiple blocks. Namely, in our example, new Blocks 710 112a and 710 112b are found concurrently as well and prolong the fork.

Hence, Bitcoin needs a way to *resolve* this situation to reestablish a network-wide consensus. To this end, full nodes (and miners) always consider the *longest chain* of blocks the valid blockchain [BMC⁺15, TS16]. However, the “longest” chain, in this case, is not defined as the chain with the most blocks but whose blocks accumulated the most expected PoW in them [BMC⁺15, Ano17], i.e., which chain is to be considered “harder” to mine overall. This value is derived from the target values of the considered blocks (`nBits` in the block header, cf. Section 2.4.1).²² Once a full node becomes aware of a branch of blocks longer than its currently considered branch, the node will revert all changes of the now-superseded branch until it reaches the forking point and then apply all changes made by the longer branch.²³ As a result, the superseded branch and its blocks become *orphaned*, i.e., Blocks 710 111a and 710 112a are not considered valid anymore. Afterward, all nodes following the same consensus rules are synchronized again and the fork is resolved.

2.4.3.2 Intentional Forks

In addition to accidental forks, there are several ways and reasons for *intentionally* forking the blockchain. In this section, we give a brief overview of intentional forks to maliciously *interfere* with Bitcoin’s normal operation and basic types of intentional forks to *update* the consensus protocol for some or all nodes.

Malicious Forks. Even though miners are incentivized to mine their blocks on top of the best blockchain tip they are currently aware of (cf. Section 2.4.1), they are theoretically free to branch off the agreed-upon main blockchain at any point. A malicious miner could use this capability to hypothetically double-spend coins by (a) having their transaction t in block B_h confirmed such that it is considered valid, (b) branching off the established blockchain at B_{h-1} with a block B'_h that does not contain t , and (c) mining blocks on top of this alternate block until this branch

²¹In this dissertation, we abstract from mining pools, i.e., nodes that collectivize their mining power to increase their competitiveness, and consider every independently mining entity a *miner*.

²²The PoW of two branches are compared via the struct `CBlockIndexWorkComparator` defined in `validation.h`; it mainly relies on the cumulative PoW stored in the attribute `nChainWork` of the class `CBlockIndex`, which is defined in `chain.h`.

²³This process is performed within the function `ActivateBestChainStep()` in `validation.cpp`.

becomes longer than the previously accepted main blockchain. However, this requires the malicious miner to solve enough PoW puzzles again to overtake the collective effort of the honest miners. The malicious miner would need to control a majority of the available hashing power to execute such an attack. Hence, this scenario is also referred to as the *51 % attack* and must be avoided as the trust in the network's consensus-finding capabilities would be undermined otherwise [TS16]. In the past, large miners had decided to limit their share of the total hash power to prevent gaining so much influence that a 51 % attack could become too realistic [Mat14].

Another malicious mining strategy that involves intentional forks is *selfish mining* [ES14]. A selfish miner tries to find new blocks as the other miners, but instead of publishing them right away, they can decide to temporarily withhold their blocks and continue extending that branch in secret. Hence, the selfish miner gains an advantage over the honest miners and can selectively disclose their “hidden” blocks depending on the overall network's activity. The key idea of selfish mining is to make the honest miners spend computation efforts in vain by mining on top of blocks that the selfish miner can supersede by tactically releasing parts of that hidden branch at the right times [ES14]. Since the honest miners are unaware of the hidden branch, they will unknowingly spend their mining efforts on extending a branch that can be orphaned at will by the selfish miner. As a result, the selfish miner can create a revenue that exceeds what they should be expected to gain based on their mining capabilities as soon as they control at least 25 % of the total hashing power [ES14].

Forks for Consensus Updates. Every full node and miner should follow the same rules for the overall network to establish consensus about the current state. However, these rules may be in need of updates over time, or disputes among the node operators about what rules to apply may emerge. For instance, the Bitcoin community heavily debated the maximum block size before settling on SegWit (cf. Section 2.3.4). In any case, required or desired updates to the consensus rules create blockchain forks, as some nodes will reject blocks that other nodes would accept. Moreover, these forks can potentially be permanent and can create new ecosystems altogether: For instance, Bitcoin Cash [Bit17e] is a permanent fork of what is now referred to as Bitcoin Core, and Bitcoin SV [Bit18a] subsequently forked off Bitcoin Cash again.

There are two basic ways of forks to roll out updates to the consensus mechanism: hard forks and soft forks. A *hard fork* is a change to the consensus rules that makes full nodes accept blocks that were previously considered invalid [Bit14d]. Hard forks require all nodes to update their rules [Bit14d], i.e., they potentially partition the network if the nodes are split regarding the update. Examples of hard forks are the creation of Bitcoin Cash and Bitcoin SV, respectively. Contrarily, a *soft fork* maintains backwards compatibility by restricting updates to only invalidating blocks that were considered valid prior to the change [Bit14e]. Hence, legacy nodes will accept all blocks also accepted by the updated nodes. With a growing share of updated nodes, the dissemination of now-invalid blocks is hindered, i.e., a sufficient adoption of the updated rules will cause that no legacy block reaches the legacy nodes anymore. For instance, P2SH (cf. Section 2.3.3.3) and SegWit (cf. Section 2.3.4) were deployed to Bitcoin this way [Bit14e]. Beyond these two fork types, we will

also discuss a recently proposed third type of fork in Chapter 5, *velvet forks*, which allow for an even more gradual deployment of consensus updates.

Rolling out network-wide consensus updates requires a prior coordination among the nodes to ensure that the designated update finds sufficient levels of support. Over the time, the Bitcoin community found increasingly refined means to facilitate this coordination. First, when P2SH was proposed, miners were encouraged to include the string `/P2SH/` in the coinbase fields of their blocks to show support; the feature was to be adopted if at least 550 of the last 1000 blocks mined up until a deadline were in favor of it [And12a]. Subsequently, the update to the coinbase field defined in BIP 34 [And12b] introduced a more fine-grained rule for gradually adopting the new blocks and shifted to using the version field in the block header to signal support of the update. Finally, BIP 9 [WTMR15] refined this approach and, among other changes, defined an updated utilization of the version field that allows miners to signal their support for multiple pending consensus updates in parallel.

In this section, we have reviewed the interactions between different nodes on the Bitcoin network required to achieve consensus about the blockchain's state. In the next section, we focus on how these interactions are enabled on a technical level by giving an introduction to Bitcoin's peer-to-peer network.

2.5 Peer-to-Peer Network

So far, we have given an overview of the most relevant concepts Bitcoin relies on to maintain its virtually immutable blockchain and thereby reach consensus about the past transactions issued by its users. We now wrap up our technical overview of Bitcoin by briefly considering its underlying peer-to-peer (P2P) network. Namely, after a short overview of the P2P network (Section 2.5.1) we discuss how nodes establish and maintain connections to each other (Section 2.5.2), how data is disseminated through the network (Section 2.5.3), and how new nodes initially obtain a copy of the blockchain to synchronize with already established nodes (Section 2.5.4).

2.5.1 Network Overview

Bitcoin is operated via a peer-to-peer (P2P) network whose nodes jointly maintain the blockchain. Namely, the nodes establish consensus by first validating any new information locally and then flooding it through the network to enable other nodes to do the same. In this section, we give a brief overview of how Bitcoin's P2P network operates²⁴ before presenting individual aspects that are relevant for the remainder of this dissertation afterward. Figure 2.10 summarizes our view on the Bitcoin network.

The Bitcoin network is a *permissionless* and *unstructured* P2P network. Hence, anybody can connect to its nodes and subsequently read blockchain data as well

²⁴Incoming messages are processed by the method `PeerManagerImpl::ProcessMessage()` defined in `net_processing.cpp`; messages are sent from multiple other methods of the same file via the method `CConnman::PushMessage()` (defined in `net.cpp`).

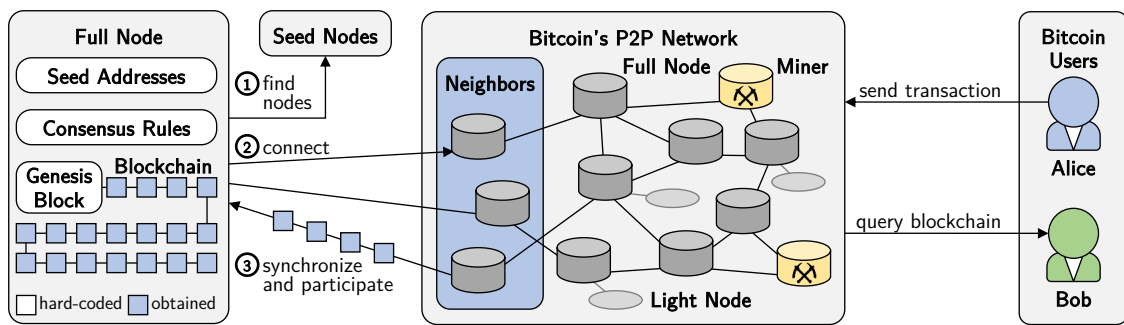


Figure 2.10 This overview shows interactions with Bitcoin's P2P network. Full nodes join the network based on hard-coded (or manually entered) seed addresses, consensus rules, and the genesis block marking the start of the blockchain. Based on this information, the full node can ① find active Bitcoin nodes by contacting the seed nodes, use this information to ② establish connections to randomly chosen neighbors, and ③ synchronize before being able to participate in the network. In this dissertation, we also consider more abstract Bitcoin users, who only send transactions into the network or read blockchain data.

as propose new data to be appended to the blockchain. Bitcoin nodes are pre-equipped with three crucial components that are hard-coded into the client: seed addresses to find other nodes, the network's consensus rules, and the genesis block as a mutual start for the blockchain for every node in the network. Based on this information, a newly joining full node can first ① find recently active nodes with the help of the seed nodes. Then, the node randomly selects and ② connects to multiple other nodes (typically eight or more [TS16]), which are referred to as the node's *neighbors*. The node has to maintain multiple connections to other nodes to limit the chances of completely relying on nodes controlled by an adversary to get information; this potential attack is referred to as an *eclipse attack* [HKZG15]. When connected to its neighbors, the node ③ synchronizes with the network and participates thereafter. First, the node requests all blocks it is currently unaware of from its neighbors. After this bootstrapping process, the node can participate in disseminating data, i.e., processing and forwarding pending transactions and newly mined blocks (cf. Section 2.4.2). Instead of running a full node, the node operator can also operate a miner, i.e., attempt to find and distribute new blocks in exchange for a reward (cf. Section 2.4.1). Finally, and orthogonally, individuals can choose to only act as Bitcoin users, i.e., merely rely on the Bitcoin network as a medium for sending transactions and retrieving data stored on the blockchain.

In the following, we characterize the different possible roles as a foundation for the remainder of this dissertation. Our characterization is loosely based on that by Antonopoulos [Ano17], but we focus on the node operator's intent. Hence, we abstract from the possibility that one node can take multiple roles.

Full Node. A full node completely validates all incoming pending transactions and newly mined blocks; furthermore, it relays valid information to its neighbors [Ano17].²⁵ For these validation purposes, a full node keeps track of an up-to-

²⁵While also other nodes may relay information, full nodes remain crucial to the Bitcoin network as only they fully validate information before relaying it. As such, we simplify our overview to assume that only full nodes relay information for the purpose of data dissemination.

date UTXO set (cf. Section 2.3.5), pending transactions in their local mempool (cf. Section 2.4.2) and, typically, they maintain a full copy of all (historic) blockchain data (cf. Section 2.5.4). As such, full nodes are the main drivers of Bitcoin’s consensus-finding mechanism, and they hence constitute the backbone of the Bitcoin network [Bit14c]. A full node is also suitable to run an independent client, i.e., send and receive payments, as the node is well-connected to disseminate payments to others and can validate received payments with its local knowledge. Keeping a full copy of the blockchain allows the full node to recover from technical issues more easily, e.g., when disk corruptions affect the derived blockchain state [Bit15b], but foremost they can provide newly joining nodes with historic data to facilitate the bootstrapping process. Nevertheless, full nodes can choose to prune historic data from their disk to conserve storage space [Bit15b]. We thus distinguish *pruning* and *non-pruning* full nodes in this dissertation; we imply full nodes to be non-pruning if not stated explicitly.

Miner. The main goal of a miner is to append new blocks to the blockchain by investing their computation resources to solve the PoW puzzle in exchange for a block reward (cf. Section 2.4.1). While miners are also inclined to validate pending transactions they plan to include in their blocks, they are only incentivized to do so to avoid having their blocks rejected by other nodes. Some miners also produce empty blocks while ignoring a backlog of pending transactions [MB15]. Hence, we view miners as complementary to full nodes and not extending upon them.

Light Node. A light node is only interested in checking for the existence of relevant transactions without the intent of contributing to the Bitcoin network [Bit18d]. Instead, a light node will trust other full nodes and make use of SPV (cf. Section 2.2.3) to confirm that an alleged payment has happened [Bit18d]. Light nodes are thus clients that depend on full nodes, since light nodes send payments to the network but have to trust full nodes to validate received payments on their behalf. Hence, this mode of operation is especially suitable for constrained devices, such as smartphones, which are used to manage payments but are incapable of running a full node [BMC⁺15].

Bitcoin User. In this dissertation, we further refer to a Bitcoin user as an abstraction of any client interested in sending and receiving payments. On a technical level, a user will operate a light node or full node with a wallet attached for managing payments. However, to simplify discussions in the following chapters, we view users as external to the Bitcoin network, and they rather rely on the Bitcoin network solely to provide a medium of consensus for their payments. Thus, we only assume that a user sends transactions to the Bitcoin network, but they may also request any valid block as well, i.e., monitor the full blockchain.

In the following, we present the crucial steps for establishing connections, disseminating data, and bootstrap nodes in Bitcoin’s P2P network.

2.5.2 Node Connections

Bitcoin nodes must first establish and then maintain connections to other nodes to participate in the P2P network. Fundamentally, each node initiatively creates *outbound* and, if enabled, accepts additional *inbound* connections for relaying blocks, transactions, and addresses from other nodes. Any node actively maintains up to eight outbound connections.²⁶ Additionally, nodes may opt to accept inbound connections from other nodes as well. Beyond that, a node can open additional special-purpose connections that are beyond the scope of this dissertation, such as short-lived “feeler” connections used to check if other nodes are currently reachable.²⁷ By default, nodes accept up to 125 connections in total.²⁸

In this section, we briefly present how joining nodes can find other nodes and how nodes establish and maintain connections with each other.

Finding Nodes. Finding any other nodes when first joining is a fundamental problem for P2P networks because of their decentralized organization [KWSW07]. Joining Bitcoin nodes have two main options for initially connecting to other nodes [Bit10b]. First, Bitcoin makes use of the Domain Name System (DNS), i.e., the Bitcoin client knows a hard-coded list of *DNS seeds*²⁹ for learning about likely active nodes [Ano17]. These DNS seeds either return a static list of reliable Bitcoin nodes or a random sample of nodes that were recently observed to be active [Ano17]. Second, the node operator can configure trusted nodes manually. For this purpose, the node operator can either specify a seed node that is only consulted initially for information about other nodes [Ano17], manually add a new connection to another node, or specify a fixed set of outbound connections, which disables any dynamic maintenance of outbound connections [Ano17]. Historically, Bitcoin had a now-removed option to look up other nodes via the Internet Relay Chat (IRC) protocol [Bit10b]. Once the joining node is connected to at least one other node using any of the methods above, it can learn about even more other nodes as the nodes actively exchange information about recently active nodes they are aware of (see “Connection Maintenance” below). If not overridden by manually connecting to a fixed set of nodes, the joining node will randomly connect to a subset of nodes it learns about to establish and maintain eight outbound connections. When a disconnected node joins the network again, that node will try to reconnect to its previously known neighbors [Ano17].

Connection Establishment. To establish a connection, two Bitcoin nodes first perform a handshake to exchange crucial information by both sending a `version` message that is acknowledged by the other party [Ano17]. Most notably, the nodes exchange information determining their compatibility, announcing special capabilities, and what block height their currently accepted blockchain tip has [Bit10c]. The nodes exchange a version number identifying the protocol version they support as well as

²⁶Outbound connections are limited via `MAX_OUTBOUND_FULL_RELAY_CONNECTIONS` in `net.h`.

²⁷All available connection types are defined in the enumeration class `ConnectionType` in `net.h`.

²⁸The number of accepted connections is defined as `DEFAULT_MAX_PEER_CONNECTIONS` in `net.h`; inbound connections are accepted via the `-listen` flag.

²⁹The DNS seeds are stored in the attribute `vSeeds` of the class `CMainParams`, which is defined in `chainparams.cpp`; currently, nine DNS seeds are hard-coded this way.

a user-agent string identifying a certain implementation of the Bitcoin client, both as defined in BIP 14 [TS11]. Furthermore, each node can advertise a list of supported *services*,³⁰ for instance, being a non-pruned full node capable of serving the full blockchain or supporting SegWit. Finally, the nodes exchange the best block height they are currently aware of to facilitate the synchronization process, i.e., the nodes get to know whether one of them can learn about new blocks from the other. After both nodes acknowledged their respective `version` message, the handshake is concluded and the connection is established.

Connection Maintenance. Since the Bitcoin network is permissionless, it must be able to cope with nodes joining and leaving at will. To this end, Bitcoin nodes (a) periodically check the liveness of their neighbors and (b) exchange information about recently active nodes to be capable of replacing stale connections if needed. If a connection becomes idle for too long, a node will ping the corresponding neighbor and assume that the neighbor disconnected if no message was received from it for 90 min [TS16]. Nodes can explicitly ask a neighbor about other nodes they know about by sending them a `getaddr` request [Bit10c, TS16]. The neighbor will then answer with an `addr` message that contains the IP addresses, ports, and advertised services of up to 1000 nodes [Bit10c, TS16]. Additionally, a node will proactively send an `addr` message containing its own address every 24 h, which gets broadcast through the network to ensure that the node becomes or remains known [Bit10b, TS16].

2.5.3 Data Dissemination

The main task of Bitcoin's P2P network is to enable its nodes to reach consensus about the system's current state, with the blockchain as its immutable record. A crucial step underlying the consensus-finding mechanisms we discussed so far in this chapter is *data dissemination*, i.e., making all (full) nodes aware of consensus-relevant updates in a decentralized manner. In this section, we briefly present how Bitcoin's full nodes exchange consensus-relevant data, i.e., transactions and blocks.

Bitcoin nodes use a flooding approach [BMC⁺15] to announce new consensus data on a network-wide scale as quickly as possible [TS16]. The most relevant information to exchange are addresses of other nodes, blocks of the blockchain, and (pending) transactions. The dissemination of addresses only facilitates the maintenance of the P2P network's structure (cf. Section 2.5.2). In contrast to this, distributing blocks and transactions is crucial for helping the nodes to establish network-wide consensus about the blockchain, as every node must be able to obtain all relevant information.

Figure 2.11 gives an overview of how Bitcoin nodes exchange information about blocks and transactions. The exchange of full blocks and transactions is handled very similarly (Figures 2.11a and 2.11b), while block headers are distributed in a slightly different way (Figure 2.11c).

When exchanging full blocks (Figure 2.11a) and transactions (Figure 2.11b), Bitcoin relies on exchanging *inventories* first. Once a node becomes aware of new information, it advertises this knowledge in an `inv` message to its neighbors [BNM⁺14],

³⁰The currently recognized services are defined in the enumeration `ServiceFlags` in `protocol.h`.

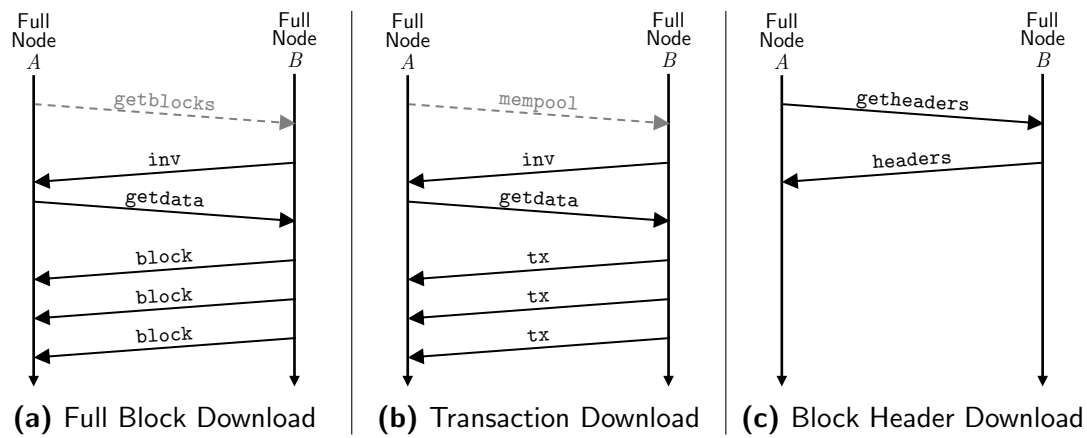


Figure 2.11 Message exchanges to disseminate (a) full blocks, (b) pending transactions, and (c) block headers, respectively, with a notation adapted from [TS16] and [Ano17]. For exchanging larger objects, i.e., full blocks and transactions, `inv` messages are used as an intermediate step to advertise available objects and allow fetching only a subset of them. Dashed requests are optional, i.e., inventories can also be advertised proactively.

which contains at most 50 000 entries [Bit10c]. Each entry announces an object type (mainly transaction or block) as well as its identifier (i.e., its SHA256 hash value) [Bit10c]. A receiving node can then match the advertised inventory against its local state and only requests the objects it does not know yet using a `getdata` message [BMC⁺15]. Finally, the node receives the requested objects individually via `block` or `tx` messages, respectively. In addition to this unsolicited advertisement of new information, a node can actively query for updates. First, the node can send a `getblocks` message to request up to 500 blocks from a neighbor (cf. Figure 2.11a) based on its currently accepted tip [Bit10c]. For instance, in older versions of the Bitcoin client, newly joining nodes used to use `getblocks` messages³¹ to request historic blocks from their neighbors during their initial synchronization (cf. Section 2.5.4). Similarly, a node can explicitly ask for the pending transactions known by a neighbor by sending a `mempool` message (cf. Figure 2.11b).

The dissemination of block headers (Figure 2.11c) follows a different pattern. Here, the node always requests a sequence of block headers by sending a `getheaders` message and then receives all requested headers in a single `headers` response [Ano17]. The layout of a `getheaders` message follows that of a `getblocks` message, but a node may request up to 2000 block headers at once (compared to the 500 full blocks that can be requested with a `getblocks` message) [Bit10c]. Instead of sending an inventory, the neighbor answers with a single `headers` response, which contains all requested block headers and the number of transactions in the respective blocks [Bit10c]. This pattern allows light nodes to obtain only the block headers as a preparation for using SPV [Ano17]: After having validated the chain of block headers, the node can use SPV to verify in a lightweight manner that a given transaction was included in a given block without having to keep track of all other transactions (cf. Section 2.2.3). However, the possibility to separate block headers and full

³¹As of Bitcoin Core v22.0, the client never sends `getblocks` messages, but it remains capable of answering such requests.

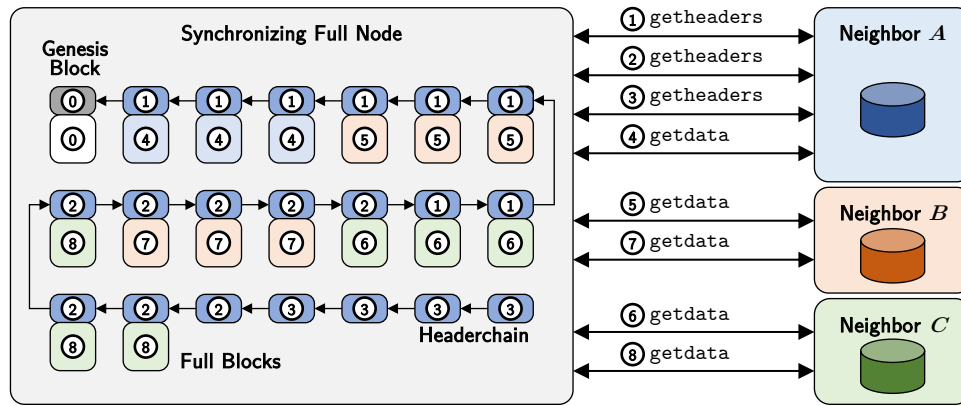


Figure 2.12 During the initial synchronization process, the joining node first downloads all block headers from one neighbor and validates them. Afterward, the node downloads the full blocks from all neighbors in parallel and applies them to build an up-to-date UTXO set.

blocks during data dissemination has since also been applied to improve the initial synchronization process for newly joining nodes, as we discuss in the next section.

2.5.4 Initial Synchronization Process

Before a newly joining full node can participate in disseminating data, and thereby contribute to the Bitcoin network, the node must first catch up with the other nodes' current view of the blockchain. In this section, we give an overview of this *initial synchronization process*, which is also referred to as *initial block download*. In this dissertation, we will also refer to this process as *bootstrapping* the node.

When a node joins the Bitcoin network for the first time, it has to establish connections to its neighbors first, as described in Section 2.5.2. The joining node learns about the length of the longest blockchain path its neighbors are aware of from the `version` messages it receives during this process. With this information, the node knows how many blocks to expect and could send `getblocks` messages to its neighbors in a round-robin fashion to receive and validate one chunk of the blockchain at a time until it reaches the full blockchain as advertised by its neighbors. In fact, this procedure used to be Bitcoin's bootstrapping process [Ano17].

However, this process was updated in 2015 with the release of Bitcoin Core v0.10.0 [Bit15a]. Now, nodes use an approach called *headers-first synchronization*, which we illustrate in Figure 2.12: Instead of building up the local blockchain copy chunk by chunk using `getblocks` requests, the node first fetches and validates the full chain of block headers, which we also refer to as the *headerchain*, before starting to request any full blocks. During the initial synchronization process,³² the node

³²This is determined in `CChainState::IsInitialBlockDownload()` in `validation.cpp`. Among other checks, the node assumes to be synchronizing as long as the best block header it accepted is older than 24h; this parameter is controlled via `DEFAULT_MAX_TIP_AGE` defined in `validation.h`.

will request the block headers from one peer exclusively.³³ After obtaining and validating the headerchain, the node is asserted that it knows the longest blockchain, i.e., the consensus reached by the Bitcoin network so far. Now, the node can start downloading the full blocks in parallel from all neighbors to gradually build up its UTXO set (cf. Section 2.3.5) by replaying each block's transactions. The node can directly send `getdata` messages for that purpose, as it already knows the block hashes from the headerchain. Once the joining node obtained an up-to-date UTXO set, it is bootstrapped successfully and can start disseminating data.

2.6 Summary

In this chapter, we gave an in-depth technical overview of Bitcoin and its crucial components to provide the necessary background for this simultaneously simple and complex distributed system. On a very high level, Bitcoin operates on a few core concepts: (a) the virtually immutable blockchain serves as a transaction ledger and medium of consensus for its users, (b) the blockchain is maintained by a permissionless P2P network of independent nodes, and (c) influencing the consensus must be hard and therefore incentivized properly. However, the implementation of this decentralized digital currency bears a much higher complexity.

For this reason, we looked at the technical realization of Bitcoin's main components in more detail. Here, we focused on the data-management perspective (cf. Chapter 1): We presented the data structures underlying Bitcoin's blockchain and especially discussed the flexibility of Bitcoin's transaction system gained by relying on the stack-based scripting language `SCRIPT`. Afterward, we presented how the individual nodes making up the Bitcoin network interact, both on a data-management level to establish consensus and on a networking level to disseminate data as a foundation to reach that consensus.

This background knowledge will accompany us throughout the next couple of chapters, where we present our technical contributions based on Bitcoin that aim to better understand and tackle the unique data-management challenges of permissionless blockchain systems.

³³The node will initially choose one neighbor for sending a `getheaders` message in the function `PeerManagerImpl::SendMessages()` in `net_processing.cpp` and only react to that node's `headers` responses during the initial synchronization process.

3

Systematic Analysis of Non-Financial Blockchain Content

In this chapter, we identify distinct challenges for data management via permissionless blockchains that stem from the content being irrevocably stored on them. To this end, we consider Bitcoin as the earliest and the structurally simplest permissionless blockchain system. Namely, we systematically analyze in the following how arbitrary content can be, and has been, engraved on the blockchains of Bitcoin Core [MHH⁺16, MHH⁺18] and its derivatives Bitcoin Cash and Bitcoin SV [MHMW24] even though Bitcoin was designed for recording only financial transactions.

We first motivate why being aware of the potential to store arbitrary contents immutably on permissionless blockchains is crucial for developers, users, and lawmakers alike (Section 3.1). Then, we recapitulate past cornerstones of blockchain content insertion and, from that, derive a simple model for the phenomenon to aid our subsequent discussions (Section 3.2). Afterward, we survey the different means available to Bitcoin users for storing non-financial content on the blockchain (Section 3.3). Based on this technical background, we discuss the benefits and risks of blockchain content insertion (Section 3.4) and then systematically analyze the utilization of different content insertion methods in Bitcoin and its derivatives (Section 3.5). Namely, we present a measurement framework that we developed for detecting content-holding transactions and extracting meta information as well as readable files for further analysis. Finally, we present related work (Section 3.6) and discuss the results and insights of our analysis before concluding this chapter (Section 3.7).

3.1 Motivation

Bitcoin [Nak08] was the first blockchain-backed digital currency and remains highly popular: Bitcoin contributes roughly 40% to the hundreds of billions of USD of

the combined market capitalization of all cryptocurrencies [Coi13] and had roughly 15 000 active and publicly reachable nodes on average during 2022 [Bit13c]. Hence, Bitcoin remains one of the most important permissionless blockchain systems.

As a consequence, researching Bitcoin remains crucial for better understanding permissionless blockchains as a whole. Namely, fundamental insights about Bitcoin and its underlying blockchain directly impact a huge user base and large monetary values. Due to its comparatively simple structure, e.g., relying on a stack-based scripting language (cf. Section 2.3.3) instead of full-fledged smart contracts as featured by Ethereum [But13, Woo14], such insights about Bitcoin are also likely to transfer to other systems implemented on top of permissionless blockchains.

Bitcoin was thus extensively analyzed in the past, and important challenges and limitations were identified this way that affect many of its core components. These challenges mainly arise as soon as malicious actors exploit the openness of permissionless blockchains (**P1**, cf. Section 1.1.2). For instance, Eyal et al. [ES14] analyzed the strategy of selfish mining, where miners can potentially improve their profit beyond their fair share by selectively withholding the blocks they mine. Alternatively, miners can launch denial-of-service (DoS) attacks to interrupt the normal operation of a blockchain, e.g., by releasing the headers of their mined blocks, but not the associated transactions [MJP⁺20]. Network-level attacks have been thoroughly investigated as well, e.g., the potential to fully separate nodes from the open Bitcoin network [HKZG15] or special attacks against nodes connected via the anonymity network Tor [BP15]. Finally, numerous analyses have shown that Bitcoin cannot live up to its initial claim to provide full anonymity to its users but instead can only provide a weaker notion of pseudonymity [MPJ⁺13, RH13, OKH13, SMZ14, FKS15, MN22].

Beyond these analyses, Bitcoin has not been rigorously investigated from a data-management perspective in the past. This is mainly due to the fact that Bitcoin was designed to provide a ledger for financial transactions. However, Bitcoin also offers means to irrevocably insert other, *non-financial content* into its blockchain. For instance, Bartoletti and Pompianu [BP17] investigated how Bitcoin users made use of the possibility to store short chunks of data in `OP_RETURN` transaction outputs (cf. Section 2.3.3.3). Furthermore, Shirriff [Shi14] collected individual examples of rich content, such as texts, images, or a PDF document, that have been embedded into transactions and have been accepted into the blockchain. Beyond this anecdotal evidence, a systematic analysis of this behavior and its consequences is still lacking.

Since anybody can essentially embed any content irrevocably, a malicious actor (**P1**) could store *illegal content* on the blockchain, e.g., child abuse imagery [BBC19]. It thus becomes imperative to systematically investigate the extent to which such unintended data-management features can be, and get, exploited and what risks such behavior bears for the operation of a blockchain system (**Q1**, cf. Section 1.2).

3.1.1 Contributions

In this chapter, we provide an initial systematic analysis of the *non-financial content* irrevocably stored on the blockchains of Bitcoin Core, Bitcoin Cash, and Bitcoin SV.

This way, we can thoroughly assess Bitcoin from a data-management perspective to (a) better understand the behavior of users of permissionless blockchains, (b) identify potential benefits motivating such behavior and threats stemming from it, and (c) provide a foundation for overcoming any identified threats. Namely, our analysis motivates our claim that blockchain content should be subject to a moderation process that is suitable for the permissionless setting (**Q1**, cf. Section 1.2).

Our analysis is three-fold, i.e., we holistically investigate the phenomenon of blockchain content insertion from a *conceptual*, a *technical*, and an *empirical* perspective.

To establish a *conceptual* foundation for further assessing our results, we first analyze the underlying problem of blockchain content insertion (Section 3.2). We begin our analysis with an informal overview of the history of past incidents and milestones regarding blockchain content insertion. This anecdotal evidence already highlights the real-world impact of the practice. Further, we derive a simple and abstract model for the insertion and distribution of blockchain content from these observations.

Afterward, we analyze and discuss the *technical* methods available to users to insert blockchain content (Section 3.3). In our analysis, we review the structures of Bitcoin’s blocks (cf. Section 2.2) and transactions (cf. Section 2.3) regarding their utility to insert non-financial blockchain content. Furthermore, we discuss programs and services available to users that facilitate blockchain content insertion.

Based on our model for blockchain content insertion and this technical background, we proceed to discuss potential benefits and risks of this practice (Section 3.4). Especially the potential threats of objectionable blockchain content have already been outlined in the past [Shi14, SLY15, MLS⁺15, AMVA17, PDC17], but they had not been thoroughly investigated before.

Finally, the main contribution of this chapter is our *empirical* analysis of blockchain content that has previously been inserted into Bitcoin-like blockchains (Section 3.5). Namely, we consider the original Bitcoin blockchain (now Bitcoin Core) as well as its permanent forks Bitcoin Cash and Bitcoin SV (cf. Section 2.4.3.2). We developed a measurement framework that allows us to scan these blockchains for non-financial content. Using this framework, we quantitatively analyze how the identified content insertion methods have been utilized over time. Furthermore, we assess readable files that we could extract via our measurement framework.

Our empirical analysis has two stages. The first stage considers the blockchain of Bitcoin Core up until the end of August 2017 [MHH⁺18]. In total, this analysis covers roughly 251 million transactions, of which 1.4% hold non-financial data of any form. We further identified more than 1600 readable files, over 99% of which contain texts or images. Our results show that, while most of this content is harmless, Bitcoin’s blockchain also contains potentially objectionable content, e.g., the depiction of a nude, young woman or hundreds of old links suggesting to point to child pornography. In the second stage of our empirical analysis, we considered the blockchains of Bitcoin Core, Bitcoin Cash, and Bitcoin SV up until July 2020 [MHMW24]. While we refrained from another in-depth investigation of files due to prior findings of child abuse imagery on the blockchain of Bitcoin SV [BBC19], we extended our high-level analysis. Here, we observe a notable shift of usage of

insertion services as Bitcoin Core and its insertion methods are largely abandoned in favor of extensively inserting content on Bitcoin SV via large `OP_RETURN` outputs.

With our analyses, we thus provide a common understanding of blockchain content insertion and especially its negative consequences. Blockchain users, developers, and researchers should take these consequences into consideration when using, developing, or improving blockchain systems. Furthermore, our results serve as a foundation for other related discussions in the upcoming chapters of this dissertation.

3.2 Problem Analysis

In this section, we analyze the problems raised by giving users the opportunity to irrevocably engrave arbitrary content into permissionless blockchains. To this end, we first provide an anecdotal overview of blockchain content insertion and its public perception (Section 3.2.1). From this overview, we then derive a simple abstract model for the insertion and distribution of blockchain content (Section 3.2.2).

3.2.1 History of Bitcoin Content Insertion

We begin our problem analysis by outlining the cornerstones of blockchain content insertion and its public perception over time. This anecdotal evidence allows us to get a first grasp of this phenomenon and its potential implications. However, even though the history of blockchain content insertion already highlights the core problem of objectionable blockchain content, the public debates cannot fully convey the technical dimension of the problem, which ultimately hinders further, neutral assessment of the consequences of blockchain content insertion and, by extension, unmoderated data management via permissionless blockchains.

The insertion of non-financial blockchain content has been an integral part of Bitcoin's history since its inception. The now-famous message "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" was engraved into the genesis block, presumably to outline a motivation to start the digital currency [Shi14]. Roughly two years later, in 2011, other non-financial content started to appear on the blockchain, as is documented by Ken Shirriff [Shi14]. For instance, an image depicting a Bitcoin logo was included into the blockchain spread over two different transactions in May 2011, Dan Kaminsky engraved a tribute to privacy advocate Len Sassaman into the blockchain in July of that year, and from August to September 2011 Catholic prayers appeared on the blockchain [Shi14].

During the same year, in June, the potential problems of allowing any Bitcoin user to persist arbitrary content on the blockchain became apparent. On the Bitcoin Forum,¹ one user claimed to have used "steganographic methods" to include presumably illegal "custom content" to Bitcoin's blockchain and that they were about to inform US police departments about the content's presence to cause "legal jeopardy" for

¹<https://bitcointalk.org>

most Bitcoin users. While this bizarre threat was met by other users mostly with sarcasm, some took note, which led to the insertion of illegal content being acknowledged as a potential weakness of Bitcoin in August 2011 [Bit10g]. Subsequently, first action to counteract content insertion could be observed: In 2013, a countermeasure to prevent users from adding arbitrary data to their transaction outputs was proposed on the mailing list for Bitcoin developers [Max13b], but this proposal was never implemented. Instead, the developers opted to introduce `OP_RETURN` transaction outputs as a semi-intended means of inserting small chunks of arbitrary content per transaction (cf. Section 2.3.3.3) with the release of Bitcoin Core v0.9.0 [Bit14a]. However, the introduction of the feature was explicitly not an endorsement of augmenting blocks with data [Bit14a]. The intention was rather to keep content-carrying transaction outputs out of the UTXO set (cf. Section 2.3.5), as `OP_RETURN` transaction outputs are provably unspendable and can thus be pruned from the UTXO set [Bit14a]. However, as we will discuss further in Section 3.5.2, the introduction of `OP_RETURN` did not prevent the utilization of other known content insertion methods that allow for larger data chunks to be stored on the blockchain.

The potential implications of uncontrolled content insertion were first acknowledged and investigated by an official body in 2015, when INTERPOL and Kaspersky Lab jointly demonstrated how blockchains could be abused to distribute chunks of malware at Black Hat Asia [KK15] and later discussed further potentials for abuse [Int15b]. Around this time, academia slowly started to acknowledge the problem as well [MLS⁺15, AMVA17]. After an initial study of blockchain content insertion [MHH⁺16], we conducted our first systematic study of this phenomenon [MHH⁺18], which we also discuss in the remainder of this chapter. Even though the community was already aware of the risks of blockchain content insertion, the presentation of our results rekindled that discussion as it reached a wider audience, including public media [Gib18, BBC18, Sha18].² While Ateniese et al. [AMVA17] already had presented the notion of a redactable blockchain in 2017, numerous academic efforts have since tackled the problem of content insertion from different perspectives, e.g., to prevent content from entering the blockchain [MHZ⁺18], to locally erase unwanted blockchain content [FHBS19], or make redactable blockchains better suited for the permissionless setting without the need to rely on trusted nodes [DMT19, MZ19, TBM⁺21, MAP⁺22a]. We will discuss the mitigation of blockchain content insertion extensively in Chapter 4.

While our 2018 study did not conclude that provably illegal content was already engraved into Bitcoin’s blockchain at that time [MHH⁺18], it strengthened the notion that single (future) instances of such content could have severe consequences (depending on the jurisdiction of individual blockchain users). This was further evidenced a year later when child abuse imagery was indeed found on the blockchain of Bitcoin SV [Mon19a], a Bitcoin fork that started in November 2018 [Bit18b] and that allows for much larger payloads in `OP_RETURN` transactions [Mon19b]. As a consequence, UK police forces started an investigation of the incident [BBC19], underpinning the potential for real-world consequences of illegal blockchain content.

²See <https://blockchain.comsys.rwth-aachen.de> for a non-exhaustive collection of articles.

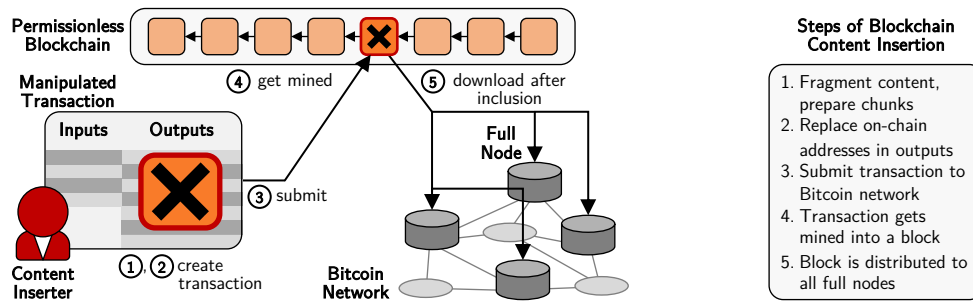


Figure 3.1 High-level overview of blockchain content insertion. Bitcoin users can create transactions that hold chunks of arbitrary data instead of valid on-chain addresses and submit those for inclusion in the blockchain. After being accepted, the block holding the transaction is distributed to all full nodes, which either store the content indefinitely or prune it while losing the capability to bootstrap other nodes in the future.

The individual incidents where illegal content has allegedly or evidently been engraved into a permissionless blockchain as well as the developing but oftentimes superficial perception of the consequences of such behavior show that further systematization of this phenomenon is necessary. Hence, we formalize blockchain content insertion and discuss its benefits and threats in the following.

3.2.2 Model for Blockchain Content Insertion and Distribution

Based on the previously reported incidents of blockchain content insertion (cf. Section 3.2.1), we derive a simple and abstract model for inserting content and how the network handles this situation in this section. Our model serves as a foundation for further assessing potential consequences of blockchain content insertion.

The openness provided by permissionless blockchains is one of their main advantages and desirable properties, as it enables transparency and decentralization: Anybody can operate a full node and thus download and independently revalidate the blockchain’s whole history (cf. Section 2.4.2). However, precisely this openness can become problematic in the light of blockchain content insertion. We have seen in the last section that users supposedly or evidently stored *objectionable* content on permissionless blockchains. If a miner adds such objectionable content to a valid block, that content gets distributed to all full nodes and is kept by all non-pruning full nodes (cf. Section 2.5.1), and, hence, becomes an irrevocable part of the blockchain.

Figure 3.1 illustrates this distribution process. We consider a Bitcoin user, the *content inserter*, who wants to store objectionable content on the blockchain. Storing content on the blockchain involves five steps in total:

The content inserter begins by preparing a content-holding transaction (Steps ① and ②). The content inserter could augment their transaction with an `OP_RETURN` transaction output containing up to 80 Byte of custom data (cf. Section 2.3.3.3). However, this limited per-transaction capacity is insufficient for encoding most content that could be viewed as objectionable, such as images. In fact, the content inserter can resort to an approach we refer to as *address manipulation* to insert

such content. We will discuss address manipulation in detail in Section 3.3.1, but, in essence, the practice works as follows: The content inserter first splits up the content into many small data chunks (Step ①) and then creates a large transaction where these chunks are inserted in place of the on-chain addresses of the output scripts (Step ②). Extracting the encoded content from a manipulated transaction then simply entails concatenating the values of the on-chain addresses again.

The user then submits the transaction for insertion in a future block (Step ③). While the transaction is effectively distributed across the network already at this stage, we focus on the case where a miner includes the transaction into a block so that the content becomes a permanent part of the blockchain. As Bitcoin was not designed with transaction manipulability in mind, the full nodes responsible for validating each transaction's correctness do not reject content-holding transactions or blocks. In fact, full nodes have no means to reliably identify manipulated on-chain addresses while transactions are pending (cf. Section 2.4.2).

A miner might then include the content-holding transaction in a block (Step ④). This way, the objectionable content can enter the blockchain and is irrevocably stored once subsequent blocks have been mined on top of the content-holding block. This content-holding block is disseminated through the Bitcoin network (cf. Section 2.5.3) and accepted by the full nodes (Step ⑤). At this point, the full nodes indefinitely store a verbatim copy of the block and the content within. Pruning full nodes can delete the raw block after processing it completely. However, in this case, they become incapable of serving other nodes the full blockchain, which impedes Bitcoin's initial synchronization process (cf. Section 2.5.4). Thus, Bitcoin requires that a sufficient number of full nodes maintain a full blockchain copy and refrain from local pruning for the system to remain operable.

Summary. Our high-level analysis of blockchain content insertion already highlights two main points. First, Bitcoin explicitly features *intended* means to insert small chunks of arbitrary content, such as `OP_RETURN` outputs, but there are also *unintended* content insertion methods that can be used to insert larger amounts of content. Second, objectionable content is distributed to all full nodes and kept indefinitely by non-pruning full nodes.

In the next section, we survey available content insertion methods in more detail before highlighting potential benefits of blockchain content insertion and discussing the threats raised by inserting objectionable content into the blockchain thereafter.

3.3 Analysis of Content Insertion Methods

In this section, we analyze how non-financial content can be inserted into Bitcoin-like blockchains on a technical level. For this purpose, we first discuss the individual low-level content insertion methods and outline their respective efficiency (Section 3.3.1). We then give an overview of higher-level protocols and services used to further facilitate content insertion (Section 3.3.2). Figure 3.2 summarizes the available insertion methods as well as the discussed higher-level insertion services.

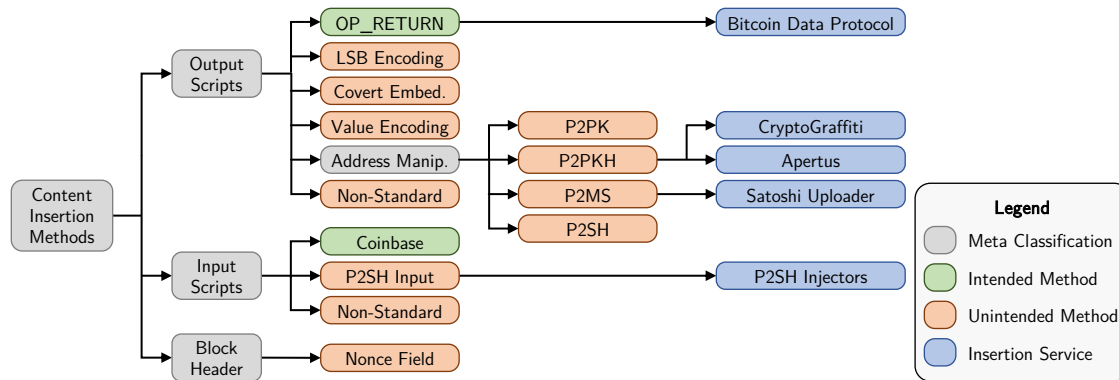


Figure 3.2 Classification of the content insertion methods available in Bitcoin. We distinguish intended and unintended methods for content insertion. Based on these methods, dedicated content insertion services emerged to provide easy-to-use interfaces for inserting content.

3.3.1 Low-Level Content Insertion Methods

Bitcoin has both *intended* but also *unintended* means of inserting blockchain content. In this section, we present the classes of content insertion methods in Bitcoin and assess their respective efficiency in terms of per-transaction capacity to insert content, whether the method burns bitcoins, and additional constraints potentially limiting the usability of a specific method. Our discussions in this dissertation focus on content insertion methods that are widely available in Bitcoin as well as related cryptocurrencies. For this reason, we exclude any impact of SegWit (cf. Section 2.3.4) from our further analyses as SegWit was a contentious discussion point; namely, Bitcoin Cash was created to provide an alternative to SegWit [KKSK19] and Bitcoin SV, another relevant fork created from Bitcoin Cash, also does not support SegWit. With this decision, we create a common ground for comparing content insertion in all of these three relevant Bitcoin-based systems in the remainder of this chapter. As a consequence, we assess the per-transaction capacity of a method based on the legacy maximum transaction size of 100 kB (cf. Section 2.3.1) instead of the transaction weight. We further do not consider the costs for inserting non-financial content due to the high volatility of market prices and transaction fees. For a cost analysis based on a snapshot of prices and fees of Bitcoin before the deployment of SegWit, we refer to our initial study on the matter [MHH⁺18]. Table 3.1 summarizes our discussion of low-level content insertion methods.

User-sided Augmentation. As we have discussed in Section 2.3.3.3, Bitcoin offers a *controlled channel* for inserting small chunks of non-financial blockchain content by augmenting transactions with an `OP_RETURN` transaction output. To recapitulate, `OP_RETURN` was introduced with Bitcoin Core v0.9.0 [Bit14a] in 2014 as a means to allow low-rate content insertion without bloating the UTXO set (cf. Section 3.4.2.1). By default, a Bitcoin user is allowed to augment their transaction with 80 Byte of payload data by adding up to one `OP_RETURN` output per transaction. However, forks of Bitcoin Core may modify this limit. For instance, Bitcoin SV does not explicitly restrict the size of `OP_RETURN` fields and, hence, the effective limit is determined by the maximum transaction size. Initially, Bitcoin SV had a maximum transaction

Insertion Method	Specific Method	Maximum Payload	Burns Coins	Insertion Constraints	Overall Efficiency
Augmentation	OP_RETURN	80 Byte*	no	One per transaction	poor
Exclusive to Miners	Coinbase	96 Byte	no	Miner only	poor
	Nonce Field	4 Byte	no	Miner only	poor
Steganography	LSB Encoding [Par18]	1 bit [†]	no	—	poor
	CDE [CYG ⁺ 22]	1 bit [†]	no	—	poor
	Value Encoding [SLY15]	3120 Byte	no	—	poor
Address Manipulation	P2PK	85 345 Byte	yes	—	high
	P2PKH	58 720 Byte	yes	—	high
	P2MS	92 625 Byte	yes	—	high
	P2SH Outputs	62 400 Byte	yes	—	high
Input Stuffing	P2SH Inputs	99 018 Byte	no	Previous transaction	high
Non-Standard Transactions	Non-Standard Outputs	99 044 Byte	depends	Colluding miner	poor
	Non-Standard Inputs	99 044 Byte	no	Previous non-standard transaction	poor

*holds for Bitcoin Core; Bitcoin Cash: 220 Byte [SGGZ19]; Bitcoin SV: 100 kB/10 MB.³

[†]the authors' designs use a capacity of 1 bit/block [Par18, CYG⁺22].

Table 3.1 Overview of the content insertion methods available in Bitcoin-based cryptocurrencies. We observe that intended methods only have a low capacity for inserting non-financial content; unintended methods offer a higher efficiency while potentially burning bitcoins.

size of 100 kB, but this limit was raised to 10 MB in December 2019.³ Third-party services started to support these large OP_RETURN fields early on [Mon19b].

Miner-exclusive Augmentation. Miners have an additional means to insert small chunks of blockchain content that is not available to every Bitcoin user. Namely, miners have the option to place roughly 100 Byte of data in the coinbase field of their successfully mined blocks, as we already mentioned in Section 2.3.3.3. Effectively, the miner has only 96 Byte to encode their content due to the 4 Byte-long encoding of the block's height at the start of the coinbase field, as mandated by BIP 34 [And12b]. We consider this content insertion method intended, as already the message in Bitcoin's genesis block was inserted via the coinbase field (cf. Section 3.2.1). Theoretically, the miner could also fix the nonce value used for solving the PoW puzzle and vary their candidate block in other ways, such as altering the block's transaction set, to insert content as well. However, this approach is tedious and only yields a low insertion capacity of 4 Byte per block. Hence, we do not further consider the nonce field in our analyses.

Steganography. Besides using OP_RETURN outputs, users could attempt to hide content in ordinary financial transactions. For instance, Partala [Par18] describes a protocol that enables a user to hide content in the least significant bits (LSBs) of their on-chain addresses. As the focus of this approach lies on establishing a covert communication channel, Partala only considers a very low capacity of encoding 1 bit of a secret message per block but argues that a higher throughput could be achieved with this method [Par18]. Similarly, Cao et al. [CYG⁺22] propose two related schemes named HC-CDE and ECDHC-CDE, which use hash chains and

³GitHub commit: <https://github.com/bitcoin-sv/bitcoin-sv/commit/a45a6ec>

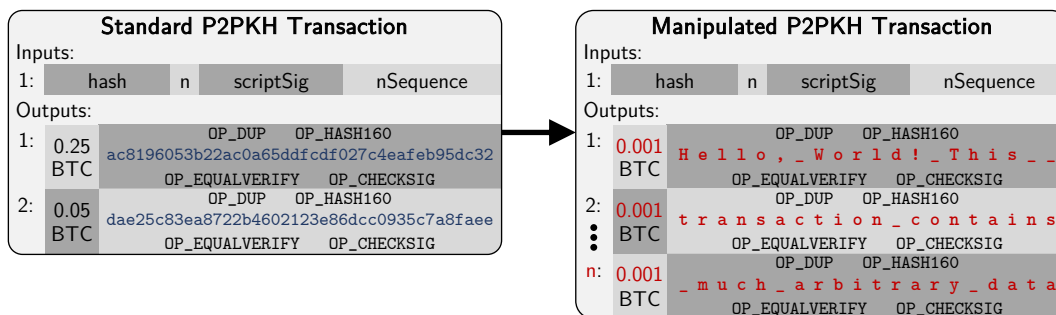


Figure 3.3 This example illustrates content insertion via address manipulation of P2PKH transaction outputs. The content inserter can create one large transaction with multiple P2PKH transaction outputs. In each output, they can insert a 20 Byte-long chunk of their content in place of the public-key hash of a recipient of the coins. This content insertion method yields a large capacity by using thousands of outputs per transaction, but each manipulated output results in permanently burning a small number of bitcoins.

Diffie-Hellman chains instead of LSB to encode one bit of information, respectively. Prior to these proposals, Sleiman et al. [SLY15] proposed to encode content via the values transferred by the outputs of a transaction. Namely, the proposed method uses arithmetic coding to hide one letter per transaction output, which yields a payload capacity of 3120 Byte per standard-sized transaction, and more efficient encoding schemes are conceivable. Even though this method does not burn currency, the content inserter needs to initially own more bitcoins than the required dust amounts per output to be able to encode different letters.

Address Manipulation. This method is especially relevant to our assumed scenario of distributing objectionable content via the blockchain (cf. Section 3.2.2) for two reasons. First, address manipulation allows for inserting content at a much larger scale than both the intended and unintended methods we discussed so far. Second, this method only uses standard patterns for encoding the content. Address manipulation refers to replacing the on-chain addresses of the output scripts for standard financial transactions with small chunks of arbitrary data. Figure 3.3 gives an overview of how address manipulation works using the example of P2PKH transaction outputs: The content inserter creates a transaction with sufficiently many P2PKH outputs, each of which transfers only a small amount (e.g., the dust amount of 546 Sat). However, the content inserter places small chunks of their content where a legitimate transaction would contain the on-chain address of a valid receiver. As we discussed in Section 2.4.2, full nodes cannot assess the legitimacy of an alleged on-chain address when validating the pending transaction; hence, they cannot check for address manipulation. Retrieving content from an address-manipulated transaction is simple as well: Except for slight variations where an additional output is created to transfer excessive funds back to the content inserter, any blockchain observer can extract and concatenate the transaction’s on-chain addresses to retrieve the content. Using P2PKH, the content inserter can split up their content into 20 Byte-long chunks and insert roughly 60 kB of content per transaction without exceeding the maximum size for standard transactions of 100 kB. The content inserter can further increase this per-transaction capacity by using P2SH, P2PK,

or P2MS output scripts instead. In terms of content insertion, those patterns have a more favorable ratio of manipulable bytes. Address manipulation burns the associated coins because the content inserter is highly unlikely to know the private key matching a manipulated public-key hash. Simultaneously, this creation of unspendable but standard-looking transaction outputs makes address manipulation a large contributor to UTXO set pollution, as we further discuss in Section 3.4.2.1. Vanity addresses or manipulated on-chain addresses that result in human-readable Bitcoin-address representations can also be considered a form of address manipulation [Rod21]; however, since these variants only manipulate a single address, we focus on the case that enables larger-scale content insertion in this dissertation.

Input Stuffing. Besides storing content in transaction outputs, the content inserter can augment input scripts as well. Intuitively, the content inserter could prepend a valid input script with a push operation inserting a chunk of the content followed by an `OP_DROP` operation without changing the evaluation of the input script. This exact approach is hardly applicable in Bitcoin because input scripts must only consist of push operations or otherwise the overall transaction is considered non-standard [Bit11d]. However, P2SH transactions enable a very similar type of content insertion due to their added flexibility (cf. Section 2.3.3.3). P2SH output scripts only contain the hash value of the actual redeem script that specifies the spendability conditions for the associated coins. While BIP 16, which defines P2SH, specifies that the redeem script should also follow the standard payment patterns we presented in Section 2.3.3.3 [And12a], the Bitcoin Core client started to accept arbitrary redeem scripts with version v0.9.2 [Ano17]. Hence, redeem scripts are now only limited by size, as push operations may push at most 520 Byte onto the stack at a time [And12a]. Sward et al. [SVS18] investigated the use of P2SH input scripts for content insertion in greater detail. Notably, content can be supplied as input for the redeem script so that P2SH input scripts can hold multiple 520 Byte-long data chunks as long as the redeem script properly processes these inputs, e.g., by simply dropping them [SVS18]. By enabling the insertion of long data chunks with only low overhead per chunk, this method can achieve even higher per-transaction capacities than address manipulation via P2MS outputs. Input stuffing via P2SH also has the additional advantages that no coins are burned and the UTXO set is not further bloated in the process. However, the content inserter must release a transaction containing the script address of the redeem script prior to being able to insert any content.

Non-Standard Transactions. Finally, the content inserter can create non-standard transactions to encode their content. Deviating from the standard patterns theoretically yields the highest possible flexibility, e.g., by maximizing the insertable content size or avoiding burning coins. However, full nodes will likely reject non-standard transactions and, hence, the content inserter has to collude with a miner to include the transaction in a block (cf. Section 2.4.2). Under this assumption, the content inserter can also insert input scripts deviating from the usual patterns if such an input script satisfies a previously accepted non-standard transaction output.

Summary. In this section, we have illustrated that there is a wide variety of available content insertion methods to irrevocably store content also on Bitcoin-like block-

chains, which predominantly focus on recording financial transactions. The most notable content insertion methods we discussed are (a) the intended augmentation of transactions with small chunks of non-financial data via `OP_RETURN` transaction outputs, (b) the miner-exclusive insertion of data via the coinbase field, and (c) large-scale content insertion via address manipulation or input stuffing.

3.3.2 Content Insertion Services

Content insertion services are auxiliary tools or online services that facilitate the insertion of blockchain content. Hence, content insertion services abstract away the technical details of the low-level insertion methods we discussed in Section 3.3.1 and provide an interface for uploading whole files, such as text documents or images, to the blockchain. In this section, we identify and discuss five conceptually different content insertion services with unique features.

Satoshi Uploader. An early content insertion service consists of two Python scripts that were persisted on Bitcoin’s blockchain in April 2013 [Shi14]. One script can be used to encode a file as a Bitcoin transaction, and the other script can extract an uploaded file again [Shi14]. We refer to both scripts as the *Satoshi Uploader*, as they are allegedly copyrighted by Satoshi Nakamoto; however, this claim remains questionable [Shi14]. The Satoshi Uploader uses 1-3-P2MS transaction outputs with uncompressed public keys (65 Byte per key, cf. Section 2.3.2.1) to embed the input file as well as the file’s CRC32 checksum into a Bitcoin transaction. The last fragment is padded with zero-bytes to fill up the last output’s public-key field. Furthermore, the Satoshi Uploader reduces the number of public keys used in the last output, and it uses a 33 Byte-long compressed public key for the last fragment if possible. Variants of the Satoshi Uploader that we discovered during our measurements further utilize other P2X outputs as well to perform address manipulation.

CryptoGraffiti. *CryptoGraffiti* [Ers16] is a web service created in 2014 that was originally intended for parsing blockchain content and subsequently was extended to also store text and files on Bitcoin’s blockchain [Che16]. Initially launched for the blockchain of Bitcoin Core, the service shifted to Bitcoin Cash in October 2017 and to Bitcoin SV in December 2018. As of the time of writing this dissertation, the upload feature has been deactivated. In the past, users of the service were instructed to send the funds required to engrave the content as well as a 10% service fee to a Bitcoin address that was then used to create the transaction [Ers16]. Similarly to the Satoshi Uploader, CryptoGraffiti relied on address manipulation, but the service used P2PKH outputs instead of P2MS outputs. Each output spent the minimally accepted dust amount of 546 Sat (cf. Section 2.4.2), and the additional profits were sent to a fixed Bitcoin address. The service allowed its users to upload files of up to 50 KiB together with optional comments.

P2SH Injectors. We refer to another class of insertion services as *P2SH Injectors*. As this name suggests, these services make use of P2SH stuffing as described in Section 3.3.1. Again, the origin of P2SH Injectors can be traced back to a Python script [LC15b], but the corresponding transactions on Bitcoin’s blockchain indicate

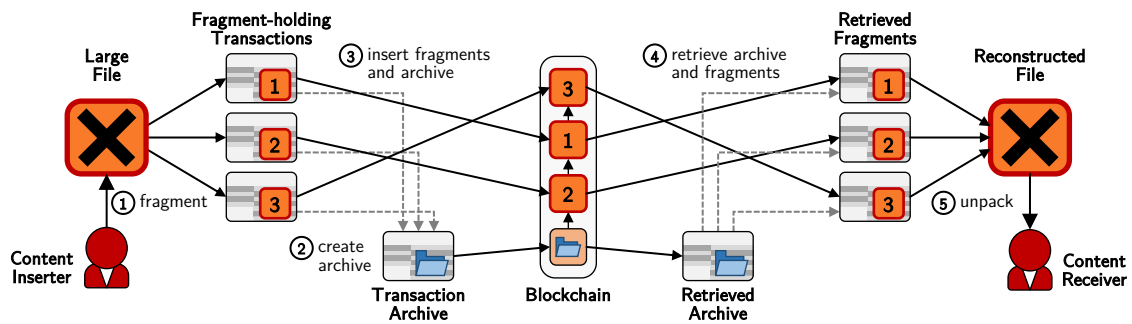


Figure 3.4 Apertus facilitates the insertion of large files by allowing the content inserter to split up the content and store it across multiple transactions. By creating an additional archive transaction holding the transaction identifiers of the individual fragments, a content receiver can obtain all fragments and concatenate them in the intended order. This way, the content size is not limited to the maximum size of a Bitcoin transaction.

slight variations in how P2SH stuffing is applied. In general, P2SH Injectors store chunks of content as a sequence of push operations within P2SH input scripts. Furthermore, the P2SH redeem script following these inputs contains and verifies hash values of these file chunks for enabling integrity validation.

Apertus. The service *Apertus* [Hug13] provides a sophisticated protocol for large-scale content insertion. To this end, Apertus is capable of splitting large files across multiple transactions and facilitate the reconstruction of such fragmented files. Figure 3.4 illustrates the general operation of Apertus. First, the content inserter ① creates multiple *fragments* of the initial file and embeds each fragment in an individual transaction using P2PKH address manipulation. Pending transactions are usually independent of each other and a user submitting multiple transactions has no direct control over the ordering of their transactions on the blockchain; due to miner preferences (cf. Section 2.4.1), the transactions may appear on the blockchain out of order, with gaps in between of them, or even scattered across multiple blocks. To mitigate this limitation, the content inserter ② creates an *archive* transaction, which holds the transaction identifiers of each fragment in their intended order. Once the fragments and the archive are ③ inserted into the blockchain, a content receiver can ④ retrieve the archive and, after parsing it, the fragments to ⑤ unpack and concatenate the fragments to reconstruct the content.

Bitcoin Data Protocol. The *Bitcoin Data Protocol (BDP)* [Unw19] is a protocol for standardized content insertion via `OP_RETURN`. As we will discuss in Section 3.5.2.2, BDP is heavily utilized with Bitcoin SV, which features exceptionally large `OP_RETURN` fields (cf. Section 3.3.1). A BDP message includes the encapsulated file’s MIME type, as well as optional indications of its encoding and file name. Other services, such as Money Button⁴ or the decentralized social network Twitch,⁵ rely on BDP and include a BDP message in the `OP_RETURN` output of their transactions. Money Button was misused to store the child abuse imagery we discussed in Section 3.2.1 on Bitcoin SV in the same month the full capacity of Bitcoin SV’s `OP_RETURN` outputs could be used [Mon19a, BBC19].

⁴<https://www.moneybutton.com>

⁵<https://twitch.app>

Summary. Various options to insert arbitrary, non-financial content into Bitcoin-like blockchains emerged over time. The spectrum here ranges from simple scripts for merely embedding content into a transaction over easy-to-use web services to sophisticated protocols to overcome space limitations imposed by consensus rules.

3.4 Benefits and Risks of Blockchain Content

After having reviewed the available methods for inserting non-financial content into Bitcoin-like blockchains, we can now assess their utility as well as potential negative consequences. We first highlight desirable use cases for persisting non-financial content on a permissionless blockchain based on the classes of available insertion methods (Section 3.4.1). Afterward, we discuss potential negative consequences of this practice both on a technical and an organizational level (Section 3.4.2).

3.4.1 Benefits of Blockchain Content Insertion

Storing content besides financial transactions on a permissionless blockchain has distinct benefits. These benefits are now systematically being seized in blockchain systems that are especially tailored to general-purpose data management, such as Ethereum with its smart contracts [But13, Woo14]. However, also the content insertion methods available in Bitcoin have valuable use cases. In the following, we discuss the use cases for the classes of content insertion methods we identified in Section 3.3. Namely, we discuss the distinguishing benefits of *user-sided transaction-level augmentation* via `OP_RETURN`, *miner-exclusive block-level augmentation* via the coinbase field, and *large-scale content insertion* via, for instance, address manipulation.

User-sided Transaction-level Augmentation. Bitcoin provides users with an intended means to augment their transactions with small chunks of non-financial content by allowing the user to include up to one `OP_RETURN` output holding up to 80 Byte payload in their transaction. This feature enables users to refer to other objects, either directly, or indirectly via cryptographic hash values, in their transactions [BP17]. Users can use this method to either bind the object specifically to their transaction or, more generally, to the position in the blockchain when a miner includes that transaction in a block. Linking real-world objects to transactions allows certifying and transferring the ownership of those objects in a transparent and accountable manner [BP17, BBP19]. Such transactions are also referred to as *colored coins* [Bit14b] or *open assets* [Cha13]. Similarly, `OP_RETURN` has been used to implement decentralized platforms for digital rights management (DRM) and the ability to store cryptographic hash values of other objects on the blockchain enables *digital notary services* [BP17, BBP19]. By storing the hash value of a file on the blockchain, a user can attest that the file existed at the time the `OP_RETURN` output was added to the blockchain [BBP19] without having to reveal the file's contents at inclusion time. Another use case is to create general-purpose non-equivocation logs by linking an external system's events to Bitcoin's blockchain via `OP_RETURN` outputs [TD17, ASNF17]. Finally, we will utilize `OP_RETURN` outputs in Chapter 6 to create

a decentralized platform for bootstrapping anonymity services. In conclusion, the user-sided augmentation of Bitcoin transactions provides a valuable tool for realizing higher-level applications that seize the properties of permissionless blockchains, such as their decentralization and immutability, to establish, e.g., transparency or accountability.

Miner-exclusive Block-level Augmentation. Via the coinbase field, miners have the intended opportunity to insert up to roughly 100 Byte of non-financial data whenever they successfully mine a block. This additional channel has further distinct benefits in comparison to the `OP_RETURN` outputs available to users. These benefits stem from a core idea behind Bitcoin’s mining process, which interprets the PoW process as giving each miner a *vote* over how the blockchain should be extended [Nak08]. In this vein, the coinbase field has been used to give miners the chance to vote for the deployment of consensus-affecting features [And12a] even though feature voting was later moved to the version field of the block header [WTMR15] (cf. Section 2.4.3). Similarly, we will utilize the coinbase field in Chapter 5 to coordinate consensus-related questions beyond Bitcoin’s mining process among the active miners. Additionally, the coinbase field has been used by miners to persist advertisements or other short messages on the blockchain [Shi14, MHH⁺16]. Finally, the coinbase field is used to extend the variability of block candidates beyond the nonce field during the mining process [Ano17]. To summarize, miner-exclusive content insertion proves valuable to facilitate consensus-related coordination beyond aspects that were already accounted for when the blockchain system was initially deployed.

Large-scale Content Insertion. Besides the intended content insertion methods, unintended methods, such as address manipulation, allow for large-scale content insertion. Even though especially this form of content insertion has been discouraged by the Bitcoin developers [Bit14a], large-scale content insertion can also provide benefits. Most notably, the content is massively replicated at many nodes. Hence, large-scale content insertion provides a means for persistently archiving historical data or censorship-resistant publication of documents, which can help protect whistleblowers or critical journalists [WRC00].

We have discussed in this section that storing non-financial data can have benefits; depending on the applied content insertion method, these benefits can range from enabling higher-level applications over advanced miner coordination to the censorship-resistant publication of documents. However, all non-pruning full nodes have to keep a copy of any non-financial content stored on the blockchain, regardless of their preference to keep or delete any specific content. As non-pruning full nodes are also vital to the Bitcoin network (cf. Section 2.5.1), we also have to consider potential negative consequences of non-financial blockchain content. Hence, we discuss the corresponding risks in the next section.

3.4.2 Negative Consequences of Blockchain Content Insertion

Despite the potential benefits of non-financial blockchain content insertion, the practice also has considerable drawbacks. In this section, we discuss the negative

consequences of such content being irreversibly stored on a permissionless blockchain. First, we highlight technical implications of blockchain content insertion (Section 3.4.2.1). Afterward, we discuss risks stemming from the nature of the inserted content, i.e., we are interested in consequences of *what* is persisted on the blockchain. To this end, we identify different categories of harmful content as a foundation for later discussions (Section 3.4.2.2) and discuss potential legal consequences of privacy-sensitive or illegal content being stored on the blockchain (Section 3.4.2.3).

3.4.2.1 Technical Implications

In this section, we discuss technical implications of blockchain content insertion in Bitcoin. Different insertion methods come with different consequences: Depending on the method, content insertion can either negatively affect the operation of full nodes by bloating the size of the blockchain and the UTXO set, or they imply financial consequences by permanently burning coins or inflicting high costs per transaction. In the following, we discuss each potential consequence in further detail.

Growing Blockchain Sizes. The append-only nature of blockchains implies that they can only grow over time as new transactions are added (**P2**, cf. Section 1.1.2). Blockchain content insertion can further aggravate this issue. Especially non-financial content being persisted on a blockchain that focuses on recording financial transactions, such as Bitcoin’s blockchain, constitutes *unwanted* content in this regard. All non-pruning full nodes have to store this content indefinitely, which uses up additional storage without being valuable for providing the intended service.

UTXO Set Bloating. Beyond forcing full nodes to store unwanted data indefinitely, the practice of manipulating on-chain addresses has additional negative effects on the operation of full nodes. When a content inserter uses address manipulation, the outputs of the resulting transaction are indistinguishable from standard financial transactions to the full nodes (cf. Section 2.3.2.2). Hence, the full nodes have to add manipulated outputs to their UTXO set, but the outputs are highly unlikely to ever be spent again. However, the full nodes use the UTXO set to improve the performance of validating pending transactions (cf. Section 2.4.2). Since this impediment is an unintended side effect of unwanted content insertion, we also refer to this effect as *UTXO set pollution*.

Burning Currency. Besides the increased storage requirements, address manipulation also has monetary implications. As UTXOs that contain manipulated on-chain addresses are effectively unspendable, so are any associated funds. Each transaction output must transfer a non-zero value that exceeds a “dust” threshold; otherwise, the transaction would be discarded (cf. Section 2.4.2). Hence, address manipulation inherently burns coins, which permanently reduces the number of available bitcoins.

Costs of Content Insertion. In addition to burning currency, the insertion of blockchain content can become costly for the content inserter. Namely, the content inserter has to pay additional transaction fees (cf. Section 2.4.1) based on the size of their transaction. The costs of inserting blockchain content varies widely and

depends on (a) the size of the content, (b) the applied insertion method, (c) the current backlog of pending transactions, which influences the per-byte transaction fees, and (d) the current exchange rate of bitcoins to fiat currency. Especially the volatility of the Bitcoin currency [Blo11b] and the varying required transaction fees [Pri20] cause the total costs of blockchain content insertion to fluctuate.

3.4.2.2 Categories of Harmful Content

Despite the potential benefits of storing also non-financial content on the blockchain we discussed in Section 3.4.1, there is also a perceived potential for harm depending on what kind of content is being stored and disseminated among full nodes [MLS⁺15, AMVA17, MHH⁺16]. In this section, we create a catalog of content that can potentially be harmful to possess for honest full node operators. We identify five categories of potentially harmful content:

Malware. One threat for Bitcoin users and full node operators is the distribution of malware over the blockchain [Int15b]. Malware can have serious consequences for end-users as it can be used by an adversary to, for example, steal personal information, or destroy sensitive documents or cause financial loss via ransomware [DDN⁺22]. Furthermore, blockchain malware can irritate users as it causes antivirus software to deny access to important blockchain files. For instance, Microsoft’s antivirus software detected a non-functional virus signature from 1987 on the blockchain, which had to be fixed manually [Wei14]. Finally, the possibility to coordinate higher-level applications (cf. Section 3.4.1) also has negative implications, as this decentralized coordination medium can be misused to operate botnets [AMLH15].

Copyright Violations. The advent of file-sharing networks established a new content distribution channel that poses a challenge for copyright holders. To suppress the distribution of pirated content, copyright holders predominantly target the users who actively offer such content to others. For instance, German law firms sue users for distributing copyright-protected content via file-sharing networks and requested fines on behalf of the copyright holders [Her15]. However, prosecutors also convicted downloaders of pirated content in some jurisdictions. For instance, France temporarily suspended users’ Internet access and subsequently switched to issuing high fines [Lee13]. As non-pruning full nodes distribute their copy of the blockchain to joining nodes, copyright-protected blockchain content could thus provoke legal disputes about copyright infringement.

Privacy Violations. Individuals can further harm their own privacy or the privacy of others by releasing sensitive personal data. Recently, jurisdictions began to tackle privacy violations actively, e.g., with the General Data Protection Regulation (GDPR) in the European Union [GDPR]. On the one hand, the GDPR sets strict boundaries for data processing (including the disclosure of personal data [GDPR, Article 4(2)]) [GDPR, Article 6]. On the other hand, service providers have the responsibility to ensure that personal data can be processed in a GDPR-compliant manner [GDPR, Article 24(1)]. Beyond general privacy-protection regulations, the release of highly personal data with malicious intent of harming others can have

serious consequences for victims [VCSA17]. Two examples of especially serious privacy invasions are the non-consensual publication of private nude photos of others (“revenge porn”) [Sch15] or fully revealing an individual’s identity to the public (“doxing”) [Dou16].

Politically Sensitive Content. Governments are generally concerned that classified information may be leaked or documents that harm national security may be distributed at a large scale, e.g., state secrets or extreme propaganda. We have discussed in Section 3.4.1 that censorship-resistant publishing is a valuable tool for whistleblowers and repressed journalists. However, distributing such content via a permissionless blockchain causes all full node operators to store content that might be declared illegal to possess or distribute by the government. Depending on the responsible jurisdiction, the intentional disclosure or any distribution of such content may be illegal. For instance, the US government usually tends to focus prosecution efforts on the intentional theft or disclosure of state secrets [Tuc16]. Contrarily, China reportedly uses broad and vague laws on state secrets to prosecute also critical journalists [Com15] or researchers [Pee05]. As another example, the Turkish government reportedly also has vaguely phrased anti-terrorism laws, which were used in the past to prosecute critical journalists [Hou16]. Without judgment of the laws applied in individual jurisdictions, these examples serve to illustrate that the acquisition and distribution of content can also become problematic for full node operators, even if that content is not clearly illegal but deemed politically sensitive.

Illegal and Condemned Content. Finally, some non-classified content is condemned and potentially prosecuted in certain jurisdictions. Religious content such as certain symbols, prayers, or sacred texts may be considered objectionable in jurisdictions that antagonize religious freedom. For example, possession of items associated with an opposed religion or publicly displayed blasphemy can be prosecuted and were sometimes even punished by death [BIH12, LB15]. While the consequences of possessing or distributing such content may strongly vary depending on the responsible jurisdiction, child pornography constitutes an almost universally condemned type of content. Namely, the possession of child pornography is illegal in at least the 178 countries [Un00b] that accepted an optional protocol to the Convention on the Rights of the Child of the United Nations from 2000 [Un00a].

Summary. There is a wide range of objectionable content that can ultimately be harmful to individuals possessing it. In contrast to social media platforms, file-sharing networks, or online storage systems, blockchain content is stored pseudonymously and irrevocably at the same time. Since operators of non-pruning full nodes download and persistently store such blockchain content (cf. Section 2.5.1), they may face consequences for possessing objectionable content added by malicious actors. Next, we further analyze this hypothetical threat by exemplarily discussing the anticipated legal consequences of privacy-sensitive and illegal content, respectively.

3.4.2.3 Potential Legal Complications

After having identified categories of potentially harmful blockchain content, we underpin the underlying threats of storing harmful content on the blockchain by exem-

plarily discussing possible legal consequences regarding *privacy-violating* and *illegal* content, respectively. Our discussions focus on German law, since Germany is a prominently represented jurisdiction within the Bitcoin network [Bit13c], but we briefly discuss other relevant jurisdictions as well. While we are not aware of specific court rulings, we highlight the potential for negative consequences for full node operators when such content is stored on the blockchain.

Privacy Violations. As we have discussed in Section 3.4.2.2, lawmakers began to create laws that have the goal of protecting users' privacy; the General Data Protection Regulation (GDPR) of the European Union [GDPR] is a good example of this development. For instance, the GDPR defines the "right to rectification" [GDPR, Article 16], i.e., a service provider has to update inaccurate data, and the "right to be forgotten" [GDPR, Article 17(1,2)], according to which service providers are obliged to erase personal data upon request and take reasonable steps to inform third parties of the requested erasure. More precisely, the GDPR defines a *controller*, who "determines the purposes and means of the processing of personal data" either as a single or joint entity [GDPR, Article 4(7)]; this controller assumes the obligations exemplified above. In Bitcoin, it becomes hard to determine who constitutes the controller as Bitcoin users, node operators, and miners all process personal data (i.e., the transfer of bitcoins to pseudonymous Bitcoin addresses) to some extent [BEHE19]. Buocz et al. argue that a collective consisting of the full node operators and the miners constitutes the controller within the meaning of the GDPR, but it remains open whether this collective acts as one entity or as individual joint controllers, which could theoretically also be held responsible individually for GDPR violations [BEHE19]. Hence, individual full node operators or miners can hypothetically face negative consequences stemming from GDPR violations.

Illegal Content. Now, we consider the possibility that clearly illegal content is irrevocably engraved into a permissionless blockchain. As we have outlined in Section 3.2.1, child abuse imagery has been added to the blockchain of Bitcoin SV in the past [Mon19a, BBC19]. In the following, we use this extreme case of illegal content as our working example for assessing potential consequences for full node operators.

When such content enters the blockchain, non-pruning full nodes obtain, keep, and distribute a copy of that content. In this regard, courts need to decide whether the content constitutes an *accessible document* as embedded on the blockchain and whether the full node operator *knowingly* possesses that content to determine whether the operator becomes prosecutable. Jurisdictions such as the USA [U.S.C., § 2256(5)], the United Kingdom [UKPGA78, Article 7(4A)(d)], or Ireland [ISB98, Article 2(1)] consider data illegal when it can be converted into a visual representation of illegal content. In this context, conversion currently only means the creation of said *visual* representation, e.g., decoding an image file for displaying it on a monitor. However, given the easy extraction of blockchain content inserted via address manipulation (cf. Section 3.3.1), we expect that extracting content from the blockchain could also be considered part of the conversion process in the future.

Some jurisdictions already consider simple manipulations of data to obtain illegal content when determining culpability. For instance, the German Criminal Code

(StGB) sanctions the possession and dissemination of illegal *documents* containing child or youth pornography [StGB, Articles 184b(1,3), 184c(1,3)]; notably, also storage devices such as hard disks that contain illegal content are considered documents [StGB, Article 11(3)]. In 2010, Hamburg’s higher regional court ruled that, considering the *intentional* consumption of illegal material within the meaning of Article 184b StGB does not depend on manually downloading such content; it rather suffices that a copy of the content can be cached by the web browser to possess the prosecutable content and the idea of ownership of such content should be applied more broadly [Bro10a].

However, to establish possession of illegal material on their computer, the alleged offender has to have *knowledge* that the content is stored on the computer because that enables them to access and view that content [PJJ17]. However, the discussion decided by Hamburg’s higher regional court in Germany revolved about child abuse material that was only present in a defendant’s web browser cache [Bro10a, PJJ17]. It remains to be decided whether unknowingly downloading prosecutable material as part of obtaining a blockchain copy is comparable to that discussion. On a technical level, the accessibility of such content is comparable due to the easy reassembly of blockchain content inserted via address manipulation (cf. Section 3.3.1). However, illegal content enters the cache of a web browser when it is loaded, e.g., for viewing it, which does not hold for blockchain content: Honest full nodes simply obtain and maintain a copy of the full blockchain and the validation of transactions only covers aspects that are technically required to validate payments (cf. Section 2.4.2), i.e., any encoded content is not accounted for in that validation process. However, knowledge can be established when the presence of illegal content is brought to the attention of a full node operator, e.g., via information campaigns such as a previous notice by INTERPOL [Int15b] or media coverage such as the news about illegal content on the blockchain of Bitcoin SV [Mon19a, BBC19]. With this knowledge, the full node operator has to immediately delete the content in order to prevent being accused of intentionally maintaining control over that content [PJJ17]. Thus, full node operators may be at risk of criminal prosecution due to illegal content being engraved into the blockchain even though they did not initially intend to view that content.

Summary. Both privacy violations and illegal content on the blockchain can have negative consequences for the crucial entities of the Bitcoin network. On the one hand, individual full node operators or miners might be held accountable for privacy violations within the meaning of the GDPR. On the other hand, full node operators may be legally obliged to remove illegal content present in their local blockchain copy, and they must not distribute such content to other nodes. However, precisely this dissemination process is crucial to the health of the Bitcoin network, as newly joining nodes depend on full nodes to serve them all historical data during the synchronization process (cf. Section 2.5.4). Hence, the presence of harmful blockchain content would imply serious risks for full node operators in particular.

In the following, we thus investigate to which extent non-financial content has been irrevocably stored on the blockchains of Bitcoin and related derivatives in the past, and what categories of content has been engraved on those blockchains.

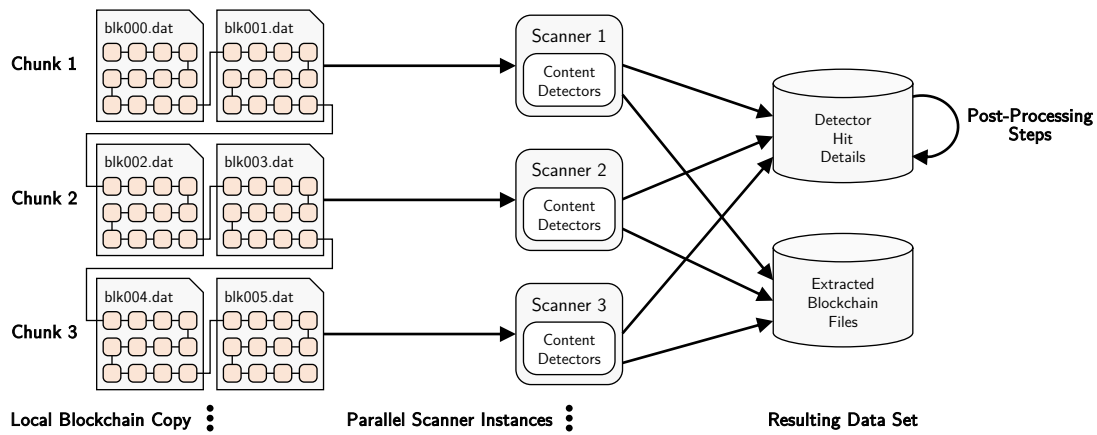


Figure 3.5 We spawn multiple, independent scanners that apply each of the considered content detectors to each transaction in their respectively assigned chunk. Each scanner stores information about each hit and, depending on the content detector, extracts the corresponding payload. The obtained data set then allows for post-processing steps to gain further insights.

3.5 Analysis of Non-Financial Content

After having analyzed the content insertion methods for Bitcoin-like cryptocurrencies as well as their associated benefits and risks, we are now interested in the extent of blockchain content insertion in the wild. In this section, we hence analyze the blockchains of Bitcoin Core and its forks Bitcoin Cash and Bitcoin SV quantitatively and qualitatively for non-financial content using a custom measurement framework.

We first give an overview of our measurement framework and the methodology we applied to analyze the considered blockchains (Section 3.5.1). Afterward, we give a quantitative overview of the utilization of content insertion methods and services over time (Section 3.5.2). Finally, we present the results of our initial investigation of files engraved into the blockchain of Bitcoin Core before discussing how file insertion has distinctly shifted since then on Bitcoin SV (Section 3.5.3).

3.5.1 Measurement Framework and Methodology

In this section, we first give an overview of the measurement framework we developed for systematically scanning Bitcoin-like blockchains (Section 3.5.1.1) before presenting the different detection mechanisms of that framework (Section 3.5.1.2). Afterward, we detail the methodology we applied based on this framework (Section 3.5.1.3) to conduct our measurements we present in the subsequent sections.

3.5.1.1 Framework Overview

For our analyses, we developed a measurement framework using the Python programming language that systematically checks each transaction of a local copy of a Bitcoin-like blockchain against a set of *content detectors*. A content detector consists of a small code segment describing the procedure to detect whether one of the

content insertion methods or services we described in Section 3.3 was used to store non-financial content on the blockchain and extract that content correspondingly. Our measurement framework is structured in a modular fashion to allow for the (a) parallelization of measurements, (b) integration of additional blockchains, and (c) extension of the framework with additional content detectors.

Figure 3.5 gives an overview of how our measurement framework parses a blockchain and stores the results. The user specifies a blockchain to analyze, the number of instances reading partitioned slices of the blockchain in parallel, and the content detectors to apply. Each instance then reads its assigned chunk of the blockchain, applies each of the specified content detectors to each transaction, and extracts relevant data for further analyses. Afterward, the results are merged again to provide a full picture of the state of non-financial content insertion in that blockchain.

Our measurement framework depends on running a synchronized non-pruning full node for the blockchain to be analyzed, i.e., our framework directly operates on the full blockchain copy maintained by full nodes. We use an adapted version of the Python library `bitcoin-blockchain-parser` [LC15a] to read individual transactions from the raw blockchain data. Our adaptations to the library enable us to parse the raw blockchain data from different Bitcoin-like blockchains (namely Bitcoin Core, Bitcoin Cash, and Bitcoin SV) and to parallelize the process. The full node software partitions the raw blockchain data into multiple files (`blocks/blk*.dat`), each of which contains multiple blocks. When spawning multiple instances of our measurement framework, each instance is assigned a subset of the raw block files and processes each transaction contained in the covered blocks individually.

For each transaction, we apply all specified content detectors to determine whether the transaction contains non-financial content and to gather relevant information. From now on, we refer to any extracted non-financial blockchain data as *content* if it has a self-contained structure, such as known file structures or readable text, or as *data* otherwise, e.g., hash values or unattributable data added via an `OP_RETURN` output. If content can be interpreted as a file, we refer to that content as a *blockchain file* accordingly. Different content detectors may flag the same transaction and thereby create multiple hits, e.g., when a coinbase transaction also makes use of address manipulation. Furthermore, individual content detectors might be heuristic and occasionally create false positives or false negatives. For instance, scanning transaction outputs for text strings may yield false positives when a legitimate public-key hash consists only of printable ASCII characters by chance. Each content detector either targets a general content insertion method, a specific content insertion service, or otherwise “suspicious” transactions that are likely to hold non-financial content. We give an overview of our implemented content detectors in the next section.

To gain additional information on transactions of interest, a full node client is running to be queried via remote procedure calls (RPC) but the client is not connected to any other node. For instance, we request the block height of identified content via RPC or whether affected transaction outputs are still unspent up to the highest block height we consider. This way, each instance can operate independently and process its assigned blocks sequentially without having to keep state.

The final output of each instance is a list of JSON objects that describe the identified hits, as well as potential meta information about the hit. We then use this additional information for final post-processing steps such as manual queries using the `jq` tool [Dol12] to gain specific insights into the generated data set.

3.5.1.2 Content Detectors

We now present the different content detectors we implemented for our measurement framework. Namely, we give a brief overview of how the content detectors scan individual transactions for non-financial content and under which conditions they flag a transaction as a hit. The content detectors presented in the following roughly correspond to the content insertion methods and services we discussed in Section 3.3. However, we omit very low-capacity insertion methods (i.e., steganographic methods or using the block header’s nonce field) and we choose to be rather restrictive with especially flexible content insertion methods (i.e., P2SH input stuffing and non-standard transactions) to keep our results more manageable. Instead, we focus on analyzing intended content insertion methods as well as large-scale content insertion, especially via dedicated services, in our measurements. Namely, we implemented simple content detectors reading payload inserted via intended means (coinbase and `OP_RETURN`) or identifying non-standard transactions; furthermore, we created detectors designed to identify content inserted via content insertion services; finally, heuristic content detectors flag text-heavy transactions or transactions that have a layout fitting the address-manipulation pattern.

Coinbase Extractor. Our first content detector investigates the coinbase transaction of every block. We keep track of the payload, its size, and the maximum number of consecutive ASCII characters in the payload. The latter enables us to identify text-holding coinbase transactions more easily during post-processing steps.

OP_RETURN Extractor. This content detector simply scans a transaction’s outputs for an output using the `OP_RETURN` operation. We extract the payload of any `OP_RETURN` output for further analysis during post-processing steps. Most notably, we subsequently parse the metadata of any BDP message we identify in the payloads of `OP_RETURN` outputs on the blockchain of Bitcoin SV (cf. Section 3.5.3.2).

Non-Standard Detector. With this content detector, we check the standardness of a transaction’s outputs (cf. Section 2.3.3.3). We flag a transaction if at least one output fails this test. While we do not fully investigate non-standard transaction outputs due to their high flexibility, we extract the series of `SCRIPT` operations of the affected output script to distinguish different types of non-standard transactions.

Satoshi Uploader Detector. Another content detector attempts to identify address-manipulated transactions that encode content like the Satoshi Uploader would do. As discussed in Section 3.3.2, the Satoshi Uploader mostly uses P2MS, but also other P2X outputs, to encode the length of the uploaded data, the data itself, and its CRC32 checksum. Hence, this content detector first concatenates all manipulable on-chain addresses from all of the transaction’s output scripts. Then, the detector

interprets the first four bytes of the resulting string as the payload's length encoded as a little-endian integer and reads the corresponding number of bytes. The detector registers a hit if the alleged payload's CRC32 checksum matches the recorded checksum in the following four bytes (also encoded as a little-endian integer).

CryptoGraffiti Detector. To detect transactions inserted via the CryptoGraffiti service, we exploit the fact that its users had to pay a fee to the service operator for their content to be engraved. This payment scheme results in CryptoGraffiti transactions forwarding the fee to a fixed Bitcoin address (`1MVpQJA7FtcDrwKC6zATkZvZcxqma4JixS`) simultaneously to engraving the submitted content via address-manipulated P2PKH transaction outputs. This content detector thus scans the transaction outputs for a payment to the fixed address and then considers all P2PKH outputs spending only tiny amounts (below 10 000 Sat per output) to be part of the content. Even though CryptoGraffiti typically only spends the dust amount of 546 Sat per output, relaxing this threshold makes the content detector more flexible.

Input-Stuffing Detector. This content detector scans for variations of the input-stuffing schemes used by P2SH Injectors (cf. Section 3.3.2). The detector inspects the redeem scripts within each P2SH input for hash operations, accepting any combination of the available hash functions (`OP_RIPEMD160`, `OP_SHA1`, `OP_SHA256`, `OP_HASH160`, or `OP_HASH256`). As soon as the redeem script performs more than one hash operation, the script deviates from the standard payment patterns (cf. Section 2.3.3.3) and we flag the whole input as input-stuffing. We further distinguish input-stuffing transactions closely following the scheme used by P2SH Injectors and any other allegedly input-stuffing transaction flagged because of using more than one hash operation in the redeem script. The detector then extracts and concatenates the data chunks from the input script that correspond to the redeem scripts' hash operations.

Apertus Detector. In comparison to the content insertion services discussed above, Apertus uses a rather complex protocol for inserting content at especially large scales by fragmenting it across multiple transactions (cf. Section 3.3.2). Apertus distinguishes content-encoding transactions and archive transactions. Archive transactions hold an ordered list of transaction identifiers pointing to the respective content-encoding transactions. As a result, the content detector for Apertus transactions is the only one that cannot integrate immediately with our framework's stateless approach. To preserve the advantages of our approach, the content detector concentrates on only finding the archive transactions and ignoring potential content-encoding transactions at first. After reading a full Apertus archive, the content detector directly requests the referenced transactions from the locally running full node via RPC. Notably, archive transactions can be stored hierarchically, i.e., one archive transaction can point to multiple fragments of another, larger archive. In this case, we unroll the nested Apertus transactions using the same RPC-based method before obtaining the engraved content. In each step of this procedure, we obtain a full Apertus message, which either holds an Apertus archive or the final content. Each message consists of multiple fields, which are separated by randomly chosen delimiters (eligible delimiters are `\`, `/`, `:`, `*`, `?`, `"`, `<`, `>`, and `|`). Any message may have a prefix before the first delimiter occurs, and all payload fields are implicitly prepended by a separate length field. Archive messages ignore this prefix

and consist only of the archive’s length and the archive, which contains the human-readable transaction identifiers separated by `\r\n` line breaks. The length field here helps to identify fragments of hierarchically composed Apertus archives. Content-encoding messages use different fields to encode meta information for the content similarly to BDP. If indicated in the prefix, the first field contains an optional digital signature. The next field holds a textual comment, which is followed by an optional filename and the content itself. Our content detector parses these messages after reassembling them and extracts the contained information accordingly.

Text Detector. With our measurement framework, we also intend to find content that was not necessarily added to the blockchain via one of the known content insertion services. This content detector aims for identifying human-readable text inserted using address manipulation. To this end, the content detector investigates each transaction output’s address-manipulable fields and flags outputs if at least 90 % of the manipulable bytes contain printable ASCII characters. Then, the address-manipulable fields of all flagged outputs of a transaction are concatenated and considered inserted text. As a result, the text detector is heuristic and can produce false positives when, for instance, a public-key hash contains sufficiently many printable ASCII characters by chance. We limit the influence of such false positives by checking with the local full node whether any of the considered outputs was already spent and marking affected transactions accordingly.

Suspicious-Transaction Detector. Finally, we implemented another heuristic approach for general-purpose content detection, i.e., not only text-based data. This content detector also focuses on address manipulation and flags transactions as “suspicious” if they have many outputs and most outputs only transfer small amounts. Specifically, the transaction must consist of at least 50 outputs spending below 100 000 Sat each. These candidate outputs must be unspent when conducting the measurement, and they must not hold more than five different amounts. These restrictions serve to filter false positives, i.e., similar-looking transactions that are likely unrelated to content insertion. For instance, “faucet services” redistribute donated bitcoins to multiple users in small amounts [RH13]. Corresponding transactions resemble address-manipulated transactions, but their outputs transfer more diverse amounts, which can be further spent by the recipients.

3.5.1.3 Methodology

In this section, we present our methodology to systematically analyze Bitcoin-like blockchains regarding content insertion. We extract our data set in two phases, namely (a) an automated filtering of (potentially) content-holding transactions using our content detectors (cf. Section 3.5.1.2) and (b) extensive automated and manual post-processing based on the filtered data.

Our analyses are based on two different measurements. Our initial study only covered the blockchain of Bitcoin Core up to Aug 31, 2017 [MHH⁺18]. We later extended that study in 2020 to cover new blocks not present in the initial measurement and also considered the blockchains of Bitcoin Cash and Bitcoin SV [MHMW24]. Namely, we

considered all three blockchains up to a block height of 640 000. While Bitcoin Core reached this block height on Jul 20, 2020, Bitcoin Cash reached it on June 17, 2020, and Bitcoin SV reached it on June 19, 2020, respectively. In all measurements, we only consider blocks exclusive to the respective blockchain (i.e., the shared prefix of Bitcoin Core and Bitcoin Cash is only considered for Bitcoin Core) and we aggregate data on a per-month basis instead of considering individual blocks.

In the following sections, we combine the results of both studies. We prioritize our extended study in our discussions but include insights from our initial study where appropriate. For example, we manually assessed readable files included via content insertion services in our initial study (cf. Section 3.5.3.1) but refrained from extending this analysis to the larger data set. The reason for this decision is two-fold. First, a heavy utilization of BDP on Bitcoin SV (cf. Section 3.5.2.2) rendered this manual approach infeasible. Second, and more notably, we refrained from accessing individual files in the extended study due to reports of child abuse imagery being engraved on the blockchain of Bitcoin SV (cf. Section 3.2.1). Instead, we resorted to analyzing the meta information of BDP transactions on Bitcoin SV. In addition to these apparent differences, our extended measurement differs in further, more subtle ways from our initial measurements. Namely, we could unveil single false positives of our initial measurement, i.e., transactions flagged as potentially content-holding by heuristic content detectors have been spent between both measurements, and our initial measurement covered at least two blocks that were later orphaned (cf. Section 2.4.3). Hence, the extended measurement describes the state of content insertion in Bitcoin-like blockchains more accurately.

Our automated filtering phase relies on the content detectors we described in Section 3.5.1.2. As a result, our results may be affected by both false positives and false negatives. False positives stem from heuristic content detectors. We aim to limit the impact of false positives by flagging any detected transaction output that is spent as a potential false positive. For instance, as described above, our extended measurement was capable of identifying additional false positives compared to our initial measurement because of this approach. False negatives are any content-holding transactions that are not detected by our content detectors. Potential sources for false negatives are transactions holding content that is obscured using steganography or that is inserted via unknown content insertion methods. Here, we use our suspicious-transaction content detector to identify additional content inserted via otherwise unknown means, but this content detector is also based on a heuristic.

We exclude identified false positives from our quantitative analysis, but we keep track of those records for potential manual inspection. Similarly, we exclude matches of our suspicious-transaction content detector due to its heuristic nature and overlaps with matches from other content detectors. Nevertheless, we discuss individual findings made via this content detector in Section 3.5.3.1.

3.5.2 Quantitative Analysis of Content Insertion

We begin our quantitative analysis of blockchain content insertion by investigating how the insertion methods we discussed in Section 3.3 have been applied in the

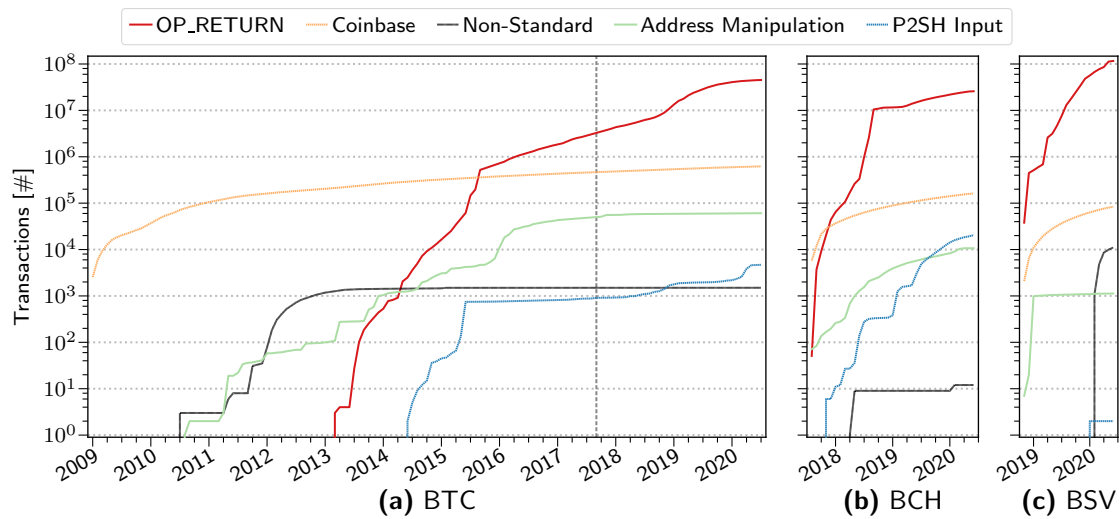


Figure 3.6 Number of transactions exclusive to the blockchains of (a) Bitcoin Core, (b) Bitcoin Cash, and (c) Bitcoin SV that were flagged by the respective content detectors. The vertical line denotes the end of our initial analysis of Bitcoin Core [MHH⁺18].

wild. To this end, we monitored the blockchains of Bitcoin Core, Bitcoin Cash, and Bitcoin SV as described in Section 3.5.1.3. In the following, we first present our results regarding the popularity of low-level insertion methods (Section 3.5.2.1) and thereafter focus on the utilization of content insertion services (Section 3.5.2.2) to assess how non-financial data enters permissionless blockchains.

3.5.2.1 Low-Level Content Insertion Methods

Data and content insertion in Bitcoin has evolved over time, transitioning from single miners augmenting the coinbase field with additional data to sophisticated services that enable storing whole files on the blockchain. In this section, we investigate the utilization of low-level content insertion methods according to our content detectors. We focus our discussions on Bitcoin Core as the oldest and most popular of the considered blockchains, and point out relevant differences of Bitcoin Cash and Bitcoin SV. Further, we point out notable developments we observed during our follow-up study [MHMW24] in contrast to our initial assessment [MHH⁺18].

Considered Blockchains. Our measurements cover the blockchains of Bitcoin Core (BTC), Bitcoin Cash (BCH), and Bitcoin SV (BSV) up to a block height of 640 000. Bitcoin Core reached this block height on Jul 20, 2020. At that point, the blockchain of BTC contained 550 220 053 transactions and had a total size of 269.03 GiB. Furthermore, we compare our findings to our initial study [MHH⁺18] where relevant, which considered 482 871 blocks holding 250 947 465 transactions and its blockchain had a total size of 122.65 GiB as of the end of August 2017. Slight variations of the numbers reported in this dissertation compared to our previous studies [MHH⁺18, MHMW24] are due to further fine-tuning of our post-processing pipeline. As BCH forked off of BTC at a block height of 478 558, we consider 161 443 blocks and 46 552 777 transactions exclusive to that blockchain with a total size of

18.85 GiB; BCH reached the target height of 640 000 blocks on Jun 17, 2020. Similarly, BSV forked off of BCH at a block height of 556 766. Hence, we consider 83 235 blocks containing 169 993 622 transactions exclusive to BSV with a total size of 82.00 GiB; BSV reached the target block height on Jun 19, 2020. The blockchain of BSV grew considerably faster than the other blockchains because BSV does not limit the block size [Bit18c]. As a result, the largest BSV block we considered has a size of 352.60 MiB, whereas the respectively imposed block-size limits yield an observed maximum block size of 2.31 MiB for BTC (post-SegWit) and 30.52 MiB for BCH. Moreover, while we observe an average block size of 0.43 MiB for BTC and 0.12 MiB for BCH, the average BSV block has a size of 1.01 MiB.

Overview. We first compare the overall utilization of content insertion methods across the three considered blockchains. Figure 3.6 shows on a logarithmic scale how many transactions using the different insertion methods we discussed in Section 3.3.1 were cumulatively included in the blockchains of BTC (Figure 3.6a), BCH (Figure 3.6b), and BSV (Figure 3.6c), respectively. In total, our content detectors flagged 45 668 592 BTC transactions with a combined payload size of 2.26 GiB. Hence, 8.3% of BTC transactions potentially contain non-financial data. Up to the cut-off of our initial study, only 1.4% of all transactions were flagged by a content detector with a total payload size of 113.29 MiB. Notably, these numbers are considerably higher for the other blockchains, with 55.9% for BCH (26 000 339 flagged transactions, 870.32 MiB payload size) and 68.4% for BSV (116 246 034 flagged transactions, 40.10 GiB payload size), respectively. Hence, content insertion had a noticeable impact across all considered blockchains. Further, `OP_RETURN` and coinbase fields constitute the vast majority of detector hits (BTC: 99.83% of all hits combined; BCH: 99.88%; BSV: 99.90%). Conversely, unintended content insertion was only responsible for comparably few detector hits (BTC: 0.17% of all hits; BCH: 0.12%; BSV: 0.10%). Even though the utilization of unintended insertion methods is overall low compared to the intended means, further investigating those methods is desirable, as even single instances of illegal content can potentially jeopardize the overall system. In general, we observe that BCH and BSV mostly show a similar, yet accelerated behavior when compared to BTC. The most notable differences in their overall utilization of content insertion methods is a relatively increased use of `OP_RETURN` outputs and varying occurrences of non-standard transactions; while we only observed very few non-standard transactions on the blockchain of BCH, BSV experienced a spike of such transactions in early 2020 that exceeds the numbers of the long-running BTC blockchain. In the following, we strive to further understand the characteristics of non-financial blockchain content.

Intended Insertion Methods. As discussed in Section 3.3.1, we consider the utilization of `OP_RETURN` outputs (for users) and the coinbase field (for miners) *intended* methods for inserting small chunks of non-financial content. While already the genesis block makes use of the coinbase field, `OP_RETURN` was officially introduced in 2014 (cf. Section 2.3.3.3). For BTC, both methods combined account for 2.24 GiB of payload (99.17% of all payloads). Notably, this dominance of payload inserted via intended means has increased compared to our initial study. Up to the end of August 2017, we could extract 95.25 MiB of `OP_RETURN` and coinbase payload data

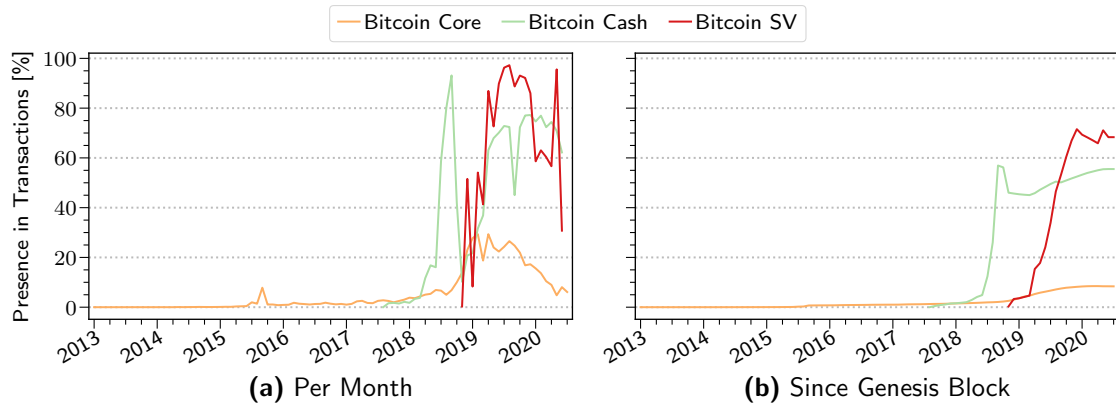


Figure 3.7 Bitcoin Cash and Bitcoin SV experience a vastly increased utilization of `OP_RETURN` outputs compared to Bitcoin Core, both **(a)** per-month and **(b)** cumulatively. As of block height 640 000, 8.2 % of Bitcoin Core transactions use `OP_RETURN`, whereas 55.5 % of Bitcoin Cash transactions and 68.3 % of Bitcoin SV transactions make use of it.

(84.08 % of all payloads at the time). Figure 3.7 further shows that `OP_RETURN` utilization gained further traction for BTC in 2018 with a burst of utilization roughly going on for the whole of 2019. On both BCH and BSV, `OP_RETURN` is used even more frequently. While we observed `OP_RETURN` outputs present in at most 29.35 % of the transactions added to BTC’s blockchain in one month (February 2019), both BCH and BSV had multiple months when at least two thirds of all transactions had an `OP_RETURN` output (BCH: 93.14 % maximum utilization; BSV: 97.24 %). As a result, 55.5 % and 68.3 % of all considered BCH and BSV transactions, respectively, contain an `OP_RETURN` output, whereas only 8.2 % of BTC transactions have one. It is to be emphasized again that BTC introduced `OP_RETURN` outputs only after four years of operation, but the discrepancy of `OP_RETURN` utilization is evident also from the per-month analysis shown in Figure 3.7a. The coinbase field is used in almost every block in some form; namely, 97.10 % of all considered BTC blocks have a non-empty coinbase field (BCH: 99.84 %; BSV: 100.00 %) but only 41.64 % of those non-empty coinbase fields contain at least 15 consecutive printable ASCII characters (BCH: 44.31 %; BSV: 52.29 %). Historically, these text strings often contain voting flags for new features (cf. Section 2.4.3.2), advertisements for the successful miner [MHH⁺16], or other short messages, e.g., prayer verses (cf. Section 3.2.1). In summary, `OP_RETURN` is widely used and well-accepted across all considered blockchains with a larger relative popularity on BCH and BSV and coinbase fields provide a regularly used communication and coordination channel for miners.

Unintended Methods. Besides the intended methods, we also identified unintended content insertion methods. Predominantly, these methods include address manipulation of P2X transaction outputs and augmenting P2SH inputs with chunks of content. Address manipulation is actively being used on all considered blockchains, whereas its utilization declined on BTC in 2018, and on BSV we only observed an initial spike of utilization during the first month of BSV’s operation that also declined afterward. Since the cut-off of our initial study after August 2017, we only observed 11 909 additional instances of address manipulation, resulting in a total of 60 718 instances and an increase of 24.4 %. In general, we can observe a shift

of using address manipulation from BTC to BCH and, briefly, BSV. Namely, BCH accumulated 10666 instances since its launch in late 2017 and BSV accumulated 967 instances in January 2019 alone, and only 160 additional instances afterward, yielding 1127 instances in total. This observation is in line with a corresponding shift of blockchains supported by the content insertion service CryptoGraffiti, which we will discuss further in Section 3.5.2.2. Compared to address manipulation, P2SH inputs are used infrequently to insert data. We investigated P2SH input scripts using our content detector for P2SH stuffing transactions (cf. Section 3.5.1.2). While primarily aimed at detecting transactions inserted via P2SH Injectors, we also found other P2SH inputs whose redeem scripts contain more than one hash operation. On BTC and BCH, we observed shorter bursts of increased utilization with almost no activity otherwise. The main burst on BTC occurred in May and June 2015, where 14.49% of the total 4686 instances of detector hits for augmented P2SH inputs were registered. In total, we found 4686 such instances on BTC, which are associated with a total payload of 9.42 MiB. During our initial study, we identified only 888 such transactions, but those already carried 8.37 MiB of payload. However, only 62 of the newly identified instances can be attributed to P2SH Injectors. On BCH, we observed several bursts over time. In total, we flagged 20472 BCH transactions (1215 attributable to P2SH Injectors). Regarding BSV, the content detector flagged only two transactions (both not attributable to a P2SH Injector).

Non-Standard Transactions. Finally, we discuss non-standard transactions, i.e., transactions containing output scripts deviating from the accepted standard patterns (cf. Section 2.3.3.3). In total, we found 1500 non-standard transactions on BTC, only twelve instances on BCH, and a comparably large number of 11059 instances on BSV. The first non-standard transactions emerged on BTC on Jul 29, 2010, where three transaction outputs excessively used repeated `OP_CHECKSIG` operations. We dedicate these transactions to an attempted DoS attack that aimed to cause high verification times for the full nodes. BTC also has several non-standard transactions that deviate slightly from the P2PKH pattern. For instance, six transactions in total append a single `OP_NOP` operation to the P2PKH pattern. Similarly, in 23 transactions from Oct 28, 2011, the on-chain address of P2PKH outputs had been replaced with `OP_0`. Between Dec 15, 2011 and Dec 19, 2012, we observe a substantial number of 986 non-standard transactions with outputs simply pushing a 32 Byte-long value onto the stack, respectively. We only found three additional non-standard transactions on BTC that were accepted into the blockchain after our initial study. Of the twelve non-standard transactions we observed on BCH, three (all from February 2020) erroneously include only a 19 Byte-long on-chain address in a P2PKH output. Similarly, four transactions deviate from the P2PKH pattern by using `OP_HASH256` instead of `OP_HASH160` and a corresponding 32 Byte-long on-chain address. On BSV, we found 11059 non-standard transactions making use of 199 different and usually relatively complex operation sequences. One of these patterns contributes to almost half of all (46%) non-standard transactions we found on BSV.

Summary. While intended insertion methods are widely used, they did not shut down unintended insertion methods entirely. Namely, the introduction of `OP_RETURN` in Bitcoin Core did not prevent address manipulation from being used subsequently. Even though unintended content insertion slowed down considerably, the risk of

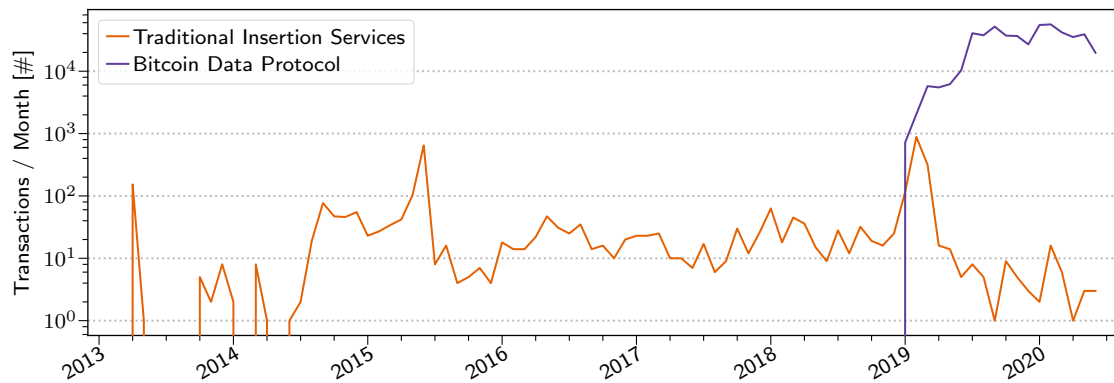


Figure 3.8 Traditional content insertion services are used continually, but infrequently. Their overall utilization further declined with the rise of the Bitcoin Data Protocol (BDP) on Bitcoin SV, which is also vastly more utilized than traditional services ever were.

single problematic instances being added to a blockchain persists. Moreover, since Bitcoin SV allows for large-scale content insertion via extended `OP_RETURN` fields, these risks are further aggravated for that blockchain. Finally, new non-standard transactions were still being added to the blockchain as of 2020.

3.5.2.2 Content Insertion Services

After having investigated the popularity of intended and unintended low-level insertion methods, we now focus on the utilization of dedicated content insertion services. In this section, we focus on general trends we observed and defer any in-depth discussion of what content has been inserted until Section 3.5.3.

Overview. In Section 3.3.2, we have identified two classes of content insertion services. The first class, comprised of the Satoshi Uploader, CryptoGraffiti, P2SH Injectors, and Apertus, use unintended insertion methods to add content to a blockchain. Contrarily, a second class of service-based content insertion makes use of Bitcoin SV’s large `OP_RETURN` outputs and the Bitcoin Data Protocol (BDP). Figure 3.8 gives an overview of how the overall utilization of both classes of services developed over time over all considered blockchains on a logarithmic scale. Our results show that service-backed content insertion occurs continually starting from June 2014, with the earliest instance in April 2013. However, even though content insertion services have been used each month afterward, traditional content insertion services were only used rarely overall. With six exceptions, all traditional content insertion services combined have only been used at most 80 times a month across all considered blockchains; we observed 3540 insertions in total. We observed insertions between April 2013 and June 2020, resulting in 40.7 insertions per month on average across all blockchains (25.5 insertions per month since BTC’s inception). In stark contrast to this rare, but steady, utilization of traditional content insertion services, BDP is being used heavily on Bitcoin SV. Since its inception in November 2018, we identified a total of 509 234 instances within the 83 235 Bitcoin SV-exclusive blocks, resulting in an average of 24 249.2 BDP insertions per month. Over the course during which we observed BDP transactions (January 2019 to June 2020), we observed 28 290.8

instances per month on average. We break down what content is being inserted using BDP in Section 3.5.3.2. In the following, we investigate how the traditional content insertion services have been utilized across all blockchains before discussing the main place where they have been used, Bitcoin Core, in more detail.

Overall Utilization of Traditional Services. Traditional insertion services have been used in low quantities, but steadily, to store blockchain files. Figure 3.9 gives a month-wise overview of how much each of the traditional insertion services has been utilized over time, both across blockchains (Figure 3.9a) and for each considered blockchain individually (Figures 3.9b–3.9d). Overall, P2SH Injectors are the largest contributor to service-based content insertion with 2011 instances. These instances are mainly concentrated in two bursts. The first burst occurred on Bitcoin Core during May and June 2015 (679 instances). During the second burst, we first observed 59 instances of P2SH Injector utilization on Bitcoin Core in December 2018 and January 2019 and then 1215 instances stored on Bitcoin Cash between January and March 2019. Next, we observed 1245 instances where CryptoGraffiti was used to insert data. In contrast to P2SH Injectors, CryptoGraffiti is being used continually since July 2014 with nine initial instances during March and April of that year. We can trace the transition from CryptoGraffiti from Bitcoin Core to Bitcoin Cash and then Bitcoin SV (cf. Section 3.3.2) in our per-blockchain breakdown of service utilization: Until September 2017, CryptoGraffiti was used exclusively on Bitcoin Core before shifting to Bitcoin Cash from October 2017 until January 2019. After this period, we only found one additional CryptoGraffiti instance on BCH in June 2019. Already in November 2018, its transition to Bitcoin SV began, where it was used infrequently, but every month, for the rest of our considered time span. Both remaining services, the Satoshi Uploader and Apertus, are used less frequently, with 167 and 117 observed instances, respectively. These services have almost exclusively been used on Bitcoin Core; namely, we only observed one instance where the Satoshi Uploader has been used on Bitcoin Cash. Overall, the utilization of traditional content insertion services decreased noticeably after March 2019, but CryptoGraffiti has actively been used throughout our considered time span to insert new content via address manipulation.

In-Depth Analysis of Bitcoin Core. We now focus on Bitcoin Core, as its blockchain experienced the most activity of traditional content insertion services. Figure 3.9b shows the utilization patterns for each service on Bitcoin Core. Since the other blockchains were launched in late 2017 and late 2018, respectively, Bitcoin Core is the only driver of the overall evolution of service-based content insertion until that point. Notably, service-based content insertion quickly faded out on Bitcoin Core and shifted to Bitcoin Cash with the exception of the 59 P2SH Injector instances in December 2018 and January 2019. As a result of Bitcoin Core’s dominance, we shift our focus now to characteristics of *what* has been inserted using content insertion services on that blockchain. Figure 3.10 shows the cumulative payload sizes of data inserted on Bitcoin Core via the respective services. In total, content insertion services account for 16.04 MiB of non-financial data on Bitcoin Core. Roughly half of this content (8.29 MiB) originates from P2SH Injectors. The remainder was mostly inserted using Apertus (3.50 MiB, 21.8 % of service-inserted data) and the Satoshi Uploader (21.4 %). Finally, CryptoGraffiti accounts for 0.82 MiB (5.1 % of

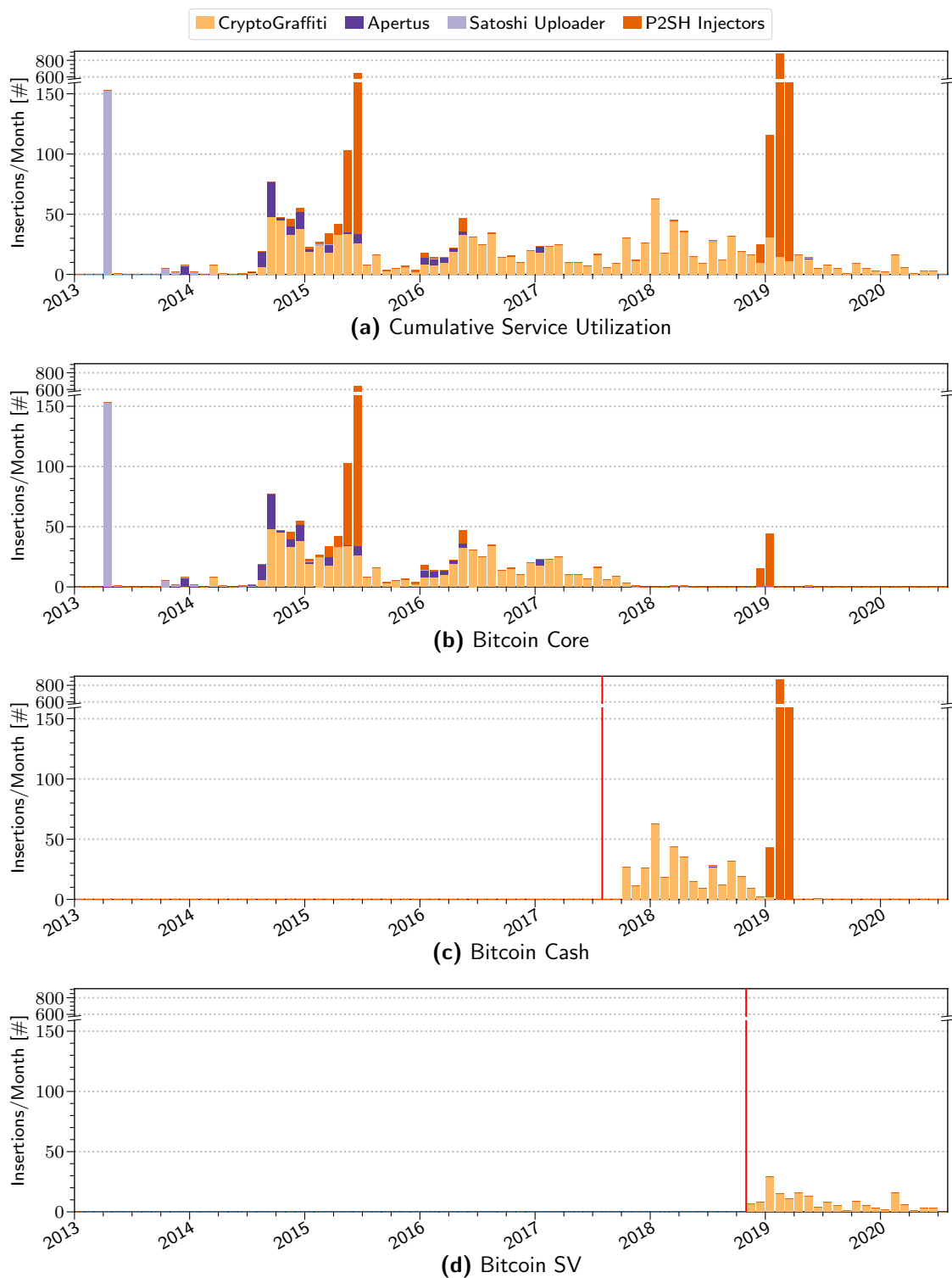


Figure 3.9 The per-month breakdown of the combined utilization of traditional content insertion services across the considered blockchains shows that they are actively being used to insert content. The largest contributors here are P2SH Injectors, which are mostly used in bursts, and the continually used CryptoGraffiti service. Further breaking down the service utilization between the considered blockchains reveals that the launches of Bitcoin Cash and Bitcoin SV (indicated by the vertical lines) respectively attracted content inserters.

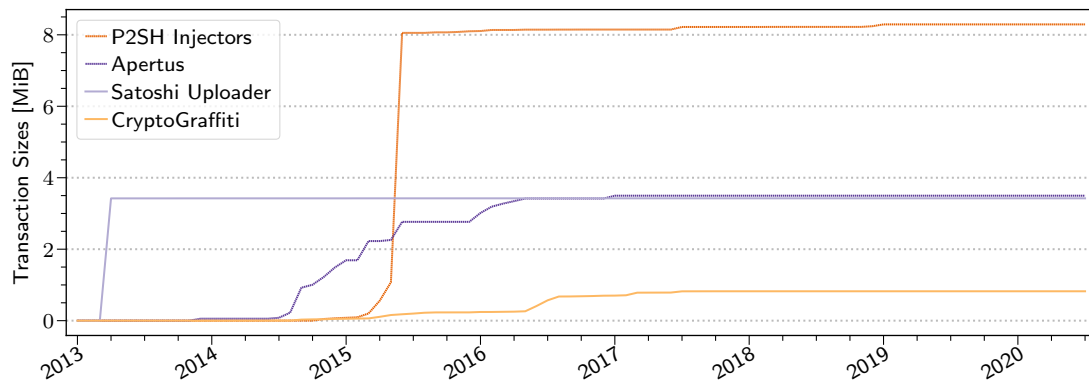


Figure 3.10 Cumulatively, P2SH Injectors were used to store the largest amount of data on the blockchain of Bitcoin Core. Even though CryptoGraffiti has been used more steadily, it has not been used to insert large amounts of data.

content related to content insertion services). Hence, Apertus and the Satoshi Uploader contribute significantly to the amount of data stemming from service-based content insertion, even though they only account for 15.7 % of all instances of service utilization on Bitcoin Core. In fact, the Satoshi Uploader was effectively used only during a brief period, as 92.2 % of all instances emerged in April 2013. During this time, the service was used to upload four archives, seven text-based files, and a PDF file. Further, Apertus is capable of fragmenting content over multiple transactions; thus, the Apertus instances are not bound to the size restrictions other content insertion services may have. In our measurements, we only counted the final archive transaction as an independent instance. Based on the median, the average Apertus file only has a size of 17.15 KiB but is spread over ten transactions (including any corresponding Apertus archives). The largest Apertus payload is 310.72 KiB large (including archives), i.e., it has over three times the size of a standard transaction, and is spread over 96 transactions. The most heavily fragmented Apertus file even spans 664 transactions. As a result, Apertus was used predominantly to store images. Contrarily, P2SH Injectors were used to back up conversations regarding the development of the Bitcoin client, especially online chat logs, forum threads, and emails, with a significant peak utilization between May and June 2015 (85.3 % of P2SH Injector instances). Finally, 90.5 % of CryptoGraffiti instances are short text messages with a median length of 69 Byte; 3.6 % of the instances contain images.

Summary. Overall, traditional content insertion services are only used infrequently except for shorter bursts. The presence of these bursts for single services indicates that a only few users may be responsible for the majority of service-inserted blockchain content. Notably, the use of traditional content insertion services shifted from Bitcoin Core to Bitcoin Cash and then Bitcoin SV, and it continued despite the availability of large `OP_RETURN` capacity on Bitcoin SV. Even though CryptoGraffiti has since deactivated its upload feature (cf. Section 3.3.2), the service was used in parallel to BDP; on Bitcoin Core, CryptoGraffiti was even used in many instances to insert shorter text messages that also fit in the smaller `OP_RETURN` fields of that blockchain. Hence, the availability of `OP_RETURN` and BDP may have effectively superseded address manipulation, but these methods did not completely shut down the utilization of services based on address manipulation.

File Type	Via Service	Unknown Origin	Overall Portion
Text	1353	54	87.07 %
Images	144	2	9.03 %
HTML	45	0	2.78 %
Source Code	7	3	0.62 %
Archive	4	0	0.25 %
Audio	2	0	0.12 %
PDF	2	0	0.12 %
Total	1557	59	100.00 %

Table 3.2 During our initial study [MHH⁺18], we retrieved 1616 readable files from the blockchain of Bitcoin Core (up to Aug 31, 2017) using our content detectors. These files were either inserted via known insertion services (1557 instances) or only detected through our suspicious-transaction content detector (59 instances).

3.5.3 Assessment of Blockchain Files

Until now, we have analyzed *how* content has been stored on Bitcoin-like blockchains in the past. In the following, we investigate *what* content is irrevocably stored on these blockchains. To this end, we first discuss findings of readable blockchain files in Bitcoin Core, which were obtained and reviewed as part of our initial study [MHH⁺18] (Section 3.5.3.1). As discussed in Section 3.3.1, Bitcoin SV enabled a distinct shift in how files can be inserted due to the availability of large `OP_RETURN` fields and BDP. Second, we thus conduct a metadata analysis of what (alleged) file types have been stored on Bitcoin SV using BDP (Section 3.5.3.2).

3.5.3.1 Early Blockchain Content

In this section, we focus on investigating readable files that are extractable from the blockchain. Specifically, we now present results from our initial analysis of the blockchain of Bitcoin Core [MHH⁺18], as we refrained from extending this analysis step in our follow-up study due to infeasibly many instances to be reviewed and ethical considerations (cf. Section 3.5.1.3). We use the term *blockchain file* to denote any content that was (a) identified by our content detectors that either scan for known content insertion services or for suspicious transactions, and (b) viewable using appropriate standard software after extraction. We reassemble fragmented files only if this is unambiguously possible, e.g., based on an Apertus archive.

Our analysis covers the first 482 871 blocks of Bitcoin Core, covering all transactions accepted up to the end of August 2017. We were able to extract and analyze 1557 blockchain files with meaningful content based on detectors targeted at identifying transactions issued by content insertion services. We further extracted 59 additional blockchain files using our suspicious-transaction detector (96.61 % text-based). Table 3.2 summarizes our findings by highlighting the different file types of the extracted files. Our results show that the vast majority of the extracted files are text-based or images (99.50 % of the extracted files).

In the following, we discuss our findings focusing on objectionable content. We manually evaluated all readable files with respect to the problematic categories we identified in Section 3.4.2.2. This analysis reveals that content related to all those categories already exists on the blockchain of Bitcoin Core. For each of these categories, we discuss the most severe examples. To protect the safety and privacy of individuals, we omit personal identifiable information and refrain from providing exact information on the location of critical content on the blockchain.

Malware. Our analysis did not directly unearth executable malware engraved on the blockchain. However, an individual non-standard transaction contains a non-malicious cross-site scripting detector. A security researcher inserted this small piece of code, which, if interpreted by an online blockchain parser, notifies the author about the vulnerability. Such malicious code could become a threat for users, as most websites offering an online blockchain parser also offer online Bitcoin accounts.

Copyright Violations. We found seven files that publish (intellectual) property and illustrate the potential to abuse Bitcoin for copyright violations. Stored are the text of a book, a copy of the original Bitcoin paper [Nak08, Shi14], and two short textual white papers. Furthermore, we found two leaked cryptographic keys, one firmware secret key [Shi14] and an RSA private key. Finally, the blockchain contains a so-called illegal prime, which encodes software that can be used to break the copy protection of DVDs [Shi14].

Privacy Violations. Bitcoin users tend to store memorable private moments on the blockchain. Namely, we extracted six wedding-related images and one image showing a group of known blockchain researchers labeled with their online pseudonyms. While these files likely constitute voluntary documentations of private or semi-private moments, such content highlights the potential for privacy breaches if such information was disclosed without consent by a third party. Furthermore, for instance, we identified 609 transactions that contain online public chat logs, emails, and forum posts discussing various aspects of Bitcoin, including topics such as money laundering. Again, disclosing *private* chat logs on the blockchain could irrevocably undermine the privacy of single users. The potential for doing harm is further exemplified by two instances of *doxing*, i.e., the complete disclosure of another individual's personal information [Dou16], we identified. This data includes phone numbers, addresses, bank accounts, passwords, and multiple online identities. As we discussed in Section 3.4.2.3, the GDPR mandates that service providers must erase such personal data upon request. However, the immutability of permissionless blockchains prevents such erasability despite the potential for serious privacy violations.

Politically Sensitive Content. Bitcoin's blockchain has been used by whistleblowers as a censorship-resistant permanent storage for leaked information. Our content detectors identified the previously reported [Shi14] backups of the WikiLeaks Cablegate data [Lyn13] and we found an online news article concerning pro-democracy demonstrations in Hong Kong in 2014 [Gra14]. As stated in Section 3.4.2.2, restrictive governments are known to prosecute the possession of such content. For example, state-critical media coverage has already put individuals in China [Com15] or Turkey [Hou16] at the risk of prosecution.

Illegal and Condemned Content. Bitcoin’s blockchain contains at least eight files with sexual content. While five files only show, describe, or link to mildly pornographic content, we consider the remaining three instances problematic: Two of them are backups of link lists to child pornography, containing 274 links to websites, 142 of which refer to Tor hidden services. The remaining instance is an image depicting mild nudity of a young woman that was later also covered by a subsequent study [GMF22]. This image was claimed to show child pornography by one user of the Bitcoin Forum (cf. Section 3.2.1), albeit this claim cannot be verified. Notably, two of the explicit images were only detected by our suspicious-transaction detector, i.e., they were not added via known insertion services.

Summary. Overall, our analysis highlights that blockchain content insertion in the wild already relates to several of the aspects we discussed in Section 3.4. While our findings do not cover obviously illegal content, and the detailed assessment of the legality of the content is out of the scope of this dissertation, the discussed blockchain files underpin the threat of potential negative consequences for blockchain users. Especially, the instances of doxing or the explicit threat of including illegal material on a blockchain indicates that blockchain-backed data management systems should not ignore the potential consequences imposed by malicious actors.

3.5.3.2 Utilization of the Bitcoin Data Protocol

In Section 3.5.2.2, we observed that the Bitcoin Data Protocol (BDP) is frequently utilized to store files on the blockchain of Bitcoin SV in much larger quantity than traditional content insertion services are being used. In this section, we further investigate what content is being inserted via BDP.

In contrast to our previous discussion of initial findings (cf. Section 3.5.3.1), we restrict ourselves to a high-level analysis of file types being inserted using BDP. This choice has two reasons. First, the number of `OP_RETURN` transactions we can attribute to carrying BDP payloads is too excessive to allow for manual investigation. Second, we refrain from a deep inspection of BDP payloads due to public reports of illegal content that has been inserted this way in the past (cf. Section 3.2.1). Instead, we exploit the structure of BDP payloads, which includes a field for declaring a MIME type for the encapsulated file, to derive the *alleged* type of content. Namely, BDP uses a fixed protocol identifier and stores binary data and meta information as individual push operations in the `OP_RETURN` field [Unw19]:

```
[ID] [payload] [mime type] [encoding (optional)] [filename (optional)] ...
```

The protocol orders the fields by decreasing significance to ensure that future extensions only append to the messages and not relocate older fields. Assuming this structure for any `OP_RETURN` payload starting with BDP’s protocol identifier, we read the MIME type where possible. When no MIME type was specified, we make assumptions about which push operation of the `OP_RETURN` script is likely to contain the payload and derive a MIME type from that data using the `python-magic` library.

As indicated by Figure 3.8, the utilization of BDP on Bitcoin SV by far exceeds that of traditional insertion services over all analyzed blockchains. For content inserters,

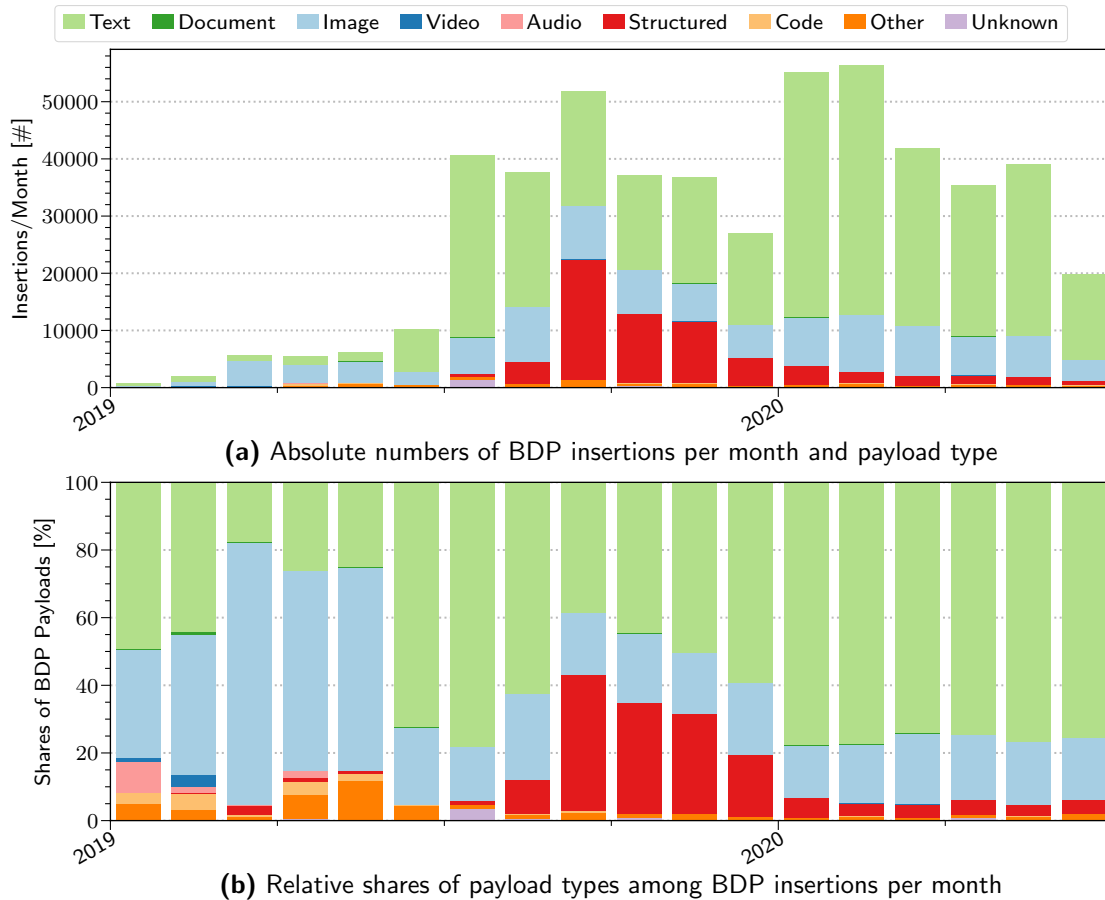


Figure 3.11 The (a) absolute and (b) relative distribution of alleged file types indicates that text-based content and images dominate the BDP data on Bitcoin SV. In late 2019, structured data, such as JSON or XML data, also contributed notably to the utilization of BDP.

this method has the advantage that Bitcoin SV's `OP_RETURN` fields are only restricted by the maximum transaction size. Specifically, the payload can take up almost 10 MB and the largest payload indeed has a size of 9.8 MB/9.3 MiB. Figure 3.11 illustrates the month-wise distribution of BDP payloads and their alleged data types in absolute numbers (Figure 3.11a) as well as their relative shares (Figure 3.11b). We observe that BDP is mostly used for storing text-based files such as plain text, HTML, or Markdown documents. Collectively, text-based payloads constitute 64.38% of all observed BDP instances. Next, 20.58% of BDP instances correspond to image data (104 820 observed instances in total). Third, structured data, such as JSON, XML, or CSV data, makes up 12.62% of the observed BDP instances. This data was inserted in a bulk between August and December 2019 but continues to constitute 4.2% of BDP instances on average since the start of 2020. Notably, BDP is also being used to insert (alleged) audio (363 instances) and video files (170 instances) as well as code (1044 instances), such as Python, JavaScript, and Shell scripts, and other documents (325 instances), for example, PDFs or Microsoft Office documents.

In summary, BDP lifted blockchain content insertion to an unprecedented level. While the majority of BDP content on Bitcoin SV is text or structured data, we observe over 100 000 alleged images, other media, and documents as well.

3.6 Related Work

Already traditional file-sharing peer-to-peer systems had to deal with the distribution of objectionable content. Furthermore, Bitcoin’s blockchain has been thoroughly analyzed since its inception; the focus of these studies lied on the privacy of Bitcoin users, but the use cases of `OP_RETURN` outputs had already drawn attention. To the best of our knowledge, our initial measurements [MHH⁺18] contributed the first systematic analysis of blockchain content also inserted via unintended means. Previously, other works had acknowledged Bitcoin’s general susceptibility to unwanted content insertion or reported individual findings. In the following, we discuss prior works related to ours and give an overview of follow-up studies. We will discuss further related work that focuses on rectifying the identified issues in Chapter 4.

Traditional Peer-to-Peer Systems. The trade-off between enabling open systems for data distribution and risking that unwanted or even illegal content is being shared is already known from traditional peer-to-peer networks. Peer-to-peer-based file-sharing protocols typically limit the spreading of objectionable *public* content by tracking the reputation of users offering files [AD01, GJA03, SUP04, ZRWW16] or assigning a reputation to files themselves [DVP⁺02, WS06]. This way, users can reject objectionable content or content from untrustworthy sources. Contrarily, distributed content stores usually encrypt *private* files before outsourcing them to other peers [CSWH01, ABC⁺03]. By storing only encrypted files, users can plausibly deny possessing any content of others and can thus obliviously store it on their hard disk. Unfortunately, these protection mechanisms are not applicable to blockchains, as their immutability prevents content from being deleted after it was accepted into a block, and the utilization of encryption cannot be enforced reliably.

Bitcoin Analyses. Bitcoin’s blockchain has been analyzed regarding different aspects by numerous studies. In a first step, multiple research groups [RS13, KPCV14, ZGH⁺15, FMR16, ZMH⁺18] studied the currency flows in Bitcoin, e.g., to perform wealth analyses. From a different line of research, several approaches focused on user privacy and investigated the identities used in Bitcoin [RH13, OKH13, MPJ⁺13, SMZ14, FKS15]. These works analyzed to which extent users can be deanonymized by clustering identities [RH13, OKH13, MPJ⁺13, SMZ14, FKS15] and augmenting these clusters with side-channel information [RH13, MPJ⁺13, SMZ14, FKS15]. Furthermore, Delgado-Segura et al. [DPNH19] analyzed a snapshot of Bitcoin’s UTXO set from October 2017. Finally, Bartoletti and Pompianu analyzed the blockchain regarding the use cases and utilization of `OP_RETURN` transaction outputs [BP17]. This work is close to ours in that it systematically analyzes non-financial information on Bitcoin’s blockchain. However, `OP_RETURN` outputs are only one of multiple means to insert non-financial content and our initial measurement [MHH⁺18] provided a first comprehensive study of the complete landscape of non-financial blockchain content.

Susceptibility to Blockchain Content. The seriousness of objectionable content stored on permissionless blockchains has been motivated by multiple works prior to ours [Shi14, SLY15, MLS⁺15, MHH⁺16, AMVA17, PDC17]. These works, however, focused on reporting individual incidents or consist of preliminary analyses

of the distribution and general utilization of content insertion. To the best of our knowledge, our contribution [MHH⁺18] gave the first comprehensive analysis of this problem space, including a categorization of objectionable content and a survey of potential risks for users if such content enters the blockchain. Other works previously analyzed attacks on Bitcoin and its ecosystem [ES14, HKZG15]. In contrast to these previously considered attacks, inserting illegal content can be instant and put full node operators at risk with the inclusion of a single transaction.

Subsequent Related Studies. During and after the publication of our initial results [MHH⁺18], further studies related to blockchain content have been conducted as well. Sward et al. [SVS18] also investigated non-financial blockchain content in parallel to our work. Their work includes a technical analysis of the available content insertion methods as well. While the authors do not consider content insertion services in their analysis, they identify more fine-grained content insertion methods based on P2SH stuffing. As a result, they report on individual content instances that were not extracted using our content detectors; however, their work does not include a systematic analysis of how the identified methods have been utilized over time.

Bartoletti et al. [BBP19] extended upon their previous work [BP17] by now considering blockchain metadata in general. This work keeps the authors' initial focus on different protocols utilizing content insertion; however, while they still predominantly consider `OP_RETURN` outputs as the means of adding blockchain content, they now also include other content insertion methods in their analysis. As such, this study provides an overview of services seizing the benefits of blockchain content insertion but only considers negative consequences on a technical level, such as UTXO set bloating. Similarly, Faisal et al. [FCS18] analyzed the utilization of `OP_RETURN` transaction outputs and highlighted additional illegitimate uses of this accepted form of content insertion in the context of ransomware.

Rodwald [Rod21] conducted an analysis of content insertion via single Bitcoin addresses. To this end, they investigated single-address manipulation via P2PKH and its pendant for Ethereum addresses. Additionally, the author mentions that short messages can further be inserted in the form of vanity addresses (cf. Section 2.3.2.2) and via address manipulation that creates a human-readable text string when parsed as a Bitcoin address. Sato et al. [SIO20] also scanned Ethereum's blockchain for relevant content and focus on files that contain illegal content or that can harm users in other ways, such as coordinating botnets. The authors report that they identified 154 files on Ethereum's blockchain, three of which contained executables classified as malware and an unspecified number of identified images contained "undesirable content." Recently, Gregoriadis et al. [GMF22] presented a new BigQuery-based framework for identifying blockchain content on Bitcoin and Ethereum. As this framework explicitly targets identifying content only via low-level insertion methods [GMF22], their analysis of Bitcoin's blockchain will yield different results compared to ours. Hence, this work is complementary to ours and their proposed framework is suitable to partially reproduce the results of our initial study.

Finally, multiple works proposed schemes for covertly inserting non-financial data into the blockchains of Bitcoin [CYG⁺22] and Ethereum [GFGC21], respectively. These means of content insertion are meant as additional private communication

channels leveraging a blockchain, i.e., the data is encrypted before insertion. The encoded content remains hidden from the full node operators not in possession of the required keys, which distinguishes covert content insertion from the plain content insertion studied in our work. Giron et al. conducted steganographic analyses of the blockchains of Bitcoin and Ethereum [GMC20, GMC21] and conclude that their forensic analyses did not reveal any hidden messages.

Summary. Permissionless blockchains were the subject of numerous analyses in the past. However, a systematic analysis of the susceptibility of these blockchains to inserting unwanted or even illegal content was lacking prior to our work, as the problem was only discussed in theory or individual findings were reported before. Our measurements closed this gap and further motivated multiple follow-up studies.

3.7 Conclusion and Future Work

In this chapter, we have systematically analyzed how users of Bitcoin-like digital currencies can irrevocably store non-financial content on the respective underlying blockchains. Namely, we demonstrated that those systems have both intended and unintended means for inserting such content, and we discussed the potential benefits and risks associated with these capabilities. Based on this initial analysis, we developed a modular measurement framework for scanning Bitcoin-like blockchains for inserted content. Using this framework, we scanned the blockchains of Bitcoin Core, Bitcoin Cash, and Bitcoin SV for non-financial content and discussed our results.

While unintended content insertion only affects a small fraction of transactions and the majority of inserted content is benign or harmless, a single instance of illegal content theoretically already has the potential for inflicting negative consequences on full node operators. Our initial analysis of files stored on Bitcoin Core’s blockchain did not yield definitive proof of such content already being present on that blockchain, but we found questionable content such as doxing, links to child pornography, or an image involving nudity and the unverifiable claim to depict a minor. The notable shift from inserting larger content via address manipulation on Bitcoin Core and Bitcoin Cash to utilizing the Bitcoin Data Protocol (BDP) in conjunction with the considerably larger `OP_RETURN` outputs offered by Bitcoin SV further aggravates this problem. Namely, this potential for negative consequences was underpinned by a first police investigation in 2019 in the UK that was started after child abuse imagery was found on the blockchain of Bitcoin SV that was inserted this way [BBC19, Mon19a].

These insights have serious implications for data-management systems operating on top of a permissionless blockchain. As these blockchain systems are open to anyone, malicious actors (**P1**, cf. Section 1.1.2) can join at any time and potentially poison the blockchain due to its immutability. While full node operators should be aware of this risk to be able to limit personal negative consequences, also the designers of such blockchain systems as well as service operators utilizing existing permissionless blockchains must consider blockchain poisoning a threat and conceptualize their platforms accordingly. For instance, `OP_RETURN` fields are widely accepted as a means

for inserting small chunks of data, and they fuel several benign Bitcoin-based applications [BP17, BBP19] (**P4**). However, exceptionally large `OP_RETURN` fields limit the implicit payload-shaping capabilities and would require further moderation of what content is being stored on the blockchain. Operators of permissioned blockchain systems can benefit as well from corresponding future developments targeted at reducing the risks associated with blockchain content insertion. Even though these blockchain systems are not necessarily open for malicious actors, they may still be required to react to content being inserted, e.g., due to the relatively recent privacy rights granted by the GDPR (cf. Section 3.4.2.3).

We identify three main directions for future work to better the understanding of blockchain content insertion and its implications. First, researchers should strive to improve the coverage of detectable content and extend it to more permissionless blockchain systems. For instance, our work aimed for a holistic overview of general insertion strategies and focused on the utilization of content insertion services, but other work has identified more fine-grained insertion methods [SVS18] and content insertion can evolve over time. The more content researchers can unveil, the better we can understand (all) applications, benign and malicious, of blockchain content insertion. Recent analyses of Ethereum [SIO20, Rod21, GFGC21, GMC21, GMF22] are promising steps toward addressing this need for further research, but the increased flexibility of smart contracts holds further challenges for fully assessing content insertion in smart contract-based blockchains. Examples of further relevant blockchain systems are privacy-focused cryptocurrencies [Sab13, Poe16] or scaling blockchains [Bru14]. Second, future work should improve the collaboration between different stakeholders monitoring blockchains for unwanted content. Ultimately, a distributed real-time monitoring system for blockchain content insertion could help researchers understand this phenomenon better and react quicker to unwanted content. However, the potential for inserting illegal content makes this otherwise desirable monitoring challenging regarding the exchange of content detectors developed by individual research groups. Other means of exchanging information about potentially problematic transactions, such as integrating fingerprint databases of known content [CTDR19], are promising developments to improve the collaboration between researchers. Third, legal scholars should further investigate the legal risks associated with running a full node for a blockchain network susceptible to illegal content. Here, our analysis (cf. Section 3.4.2.3) provides a first insight into potential negative consequences, but a more thorough discussion has to follow to cover this problem space in more detail and for other jurisdictions.

In summary, our work constituted the first systematic analysis of blockchain content insertion to the best of our knowledge. As a result, our findings sparked major international media attention (cf. Section 3.2.1) and several follow-up studies (cf. Section 3.6), and provided a new foundation for the much-needed discussion of the risks associated with unwanted blockchain content insertion.

After analyzing the problem of blockchain content insertion in this chapter, we dedicate the next chapter to prevention and mitigation strategies to alleviate the negative consequences of this practice for the blockchain-backed data management.

4

Mitigation of Unwanted Blockchain Content

After having identified the serious threats arising when malicious actors misuse blockchain-based data-management systems in the previous chapter, we now present strategies to *mitigate* the possible negative consequences in such systems. In this chapter, we investigate two orthogonal and combinable strategies for counteracting unwanted content in Bitcoin-like blockchain systems, i.e., *preventing* unwanted content from entering the blockchain and retrospectively *removing* such content.

In the following, we first motivate the need for mitigation schemes and analyze challenges and requirements for their design (Section 4.1). Following this analysis, we first present simple prevention strategies [MHZ⁺18] that can provide a first line of defense against unwanted blockchain content with only minimal adjustments to Bitcoin’s consensus rules (Section 4.2). Afterward, we present RedactChain [MAP⁺22a], a moderation framework for retrospectively removing such content from the blockchain that can be implemented with further adjustments to the consensus rules (Section 4.3). Finally, we conclude this chapter with a discussion of these orthogonal strategies to mitigate unwanted blockchain content (Section 4.4).

4.1 Motivation

In Chapter 3, we highlighted the serious consequences that unwanted blockchain content inserted by malicious actors (**P1**, cf. Section 1.1.2) can provoke for the full nodes constituting a permissionless blockchain’s backbone network. These consequences range from technical nuisances, such as impeded performance due to UTXO set pollution (cf. Section 3.4.2.1), to potential legal culpability due to illegal content being persistently stored on the blockchain and subsequently being disseminated by the nodes.

These threats raise the question of how to *protect* permissionless blockchain systems and its nodes from those consequences (**Q1**, cf. Section 1.2). Naïvely, full node operators could delete any transaction they identify to contain content they object to. However, this approach has two drawbacks. First, unilaterally deleting transactions makes a full node dependent on the validation by other full nodes if outputs of a deleted transaction get referenced in the future [FHBS19]. Moreover, if too many nodes decide to delete a specific transaction, the network may not be able to properly validate future transactions referencing deleted ones anymore [FHBS19]. Second, the public verifiability of permissionless blockchains is precisely achieved by having many redundant sources that can provide the full blockchain history to help bootstrap newly joining nodes that only know the hard-coded genesis block and the validation rules beforehand. Therefore, full nodes cannot bootstrap new nodes if they simply delete individual transactions from their local blockchain copy.

Both drawbacks imply that the full nodes of a permissionless blockchain system have to *coordinate* their activities to defend themselves from negative consequences (**P3**). In this regard, the full nodes have two basic strategies at their disposal: They can either aim to jointly *prevent* unwanted content from entering the blockchain in the first place, or they can *retrospectively remove* identified unwanted content. Preventing unwanted content insertion has the advantage that the full nodes can uphold all characteristic properties of permissionless blockchains (cf. Section 1.1.1), especially its immutability, by adding additional validation rules. As such, existing blockchain systems could easily be retrofitted with prevention schemes (**Q2**). Unfortunately, we already observed that content inserters have a multitude of insertion methods at their disposal (cf. Section 3.3), which potentially enable them to evade any prevention efforts. In fact, we argue in the following (Section 4.1.1.2) that *full prevention of content insertion cannot be achieved*. Hence, investigating approaches to retrospectively remove unwanted content, and therefore challenge the blockchain’s immutability, is crucial as well. Numerous corresponding blockchain designs have been proposed in the past. However, the main approaches to realize *redactable blockchains* fall into one of two categories, both of which have individual shortcomings: On the one hand, trust-based approaches empower either a single entity or a consortium of few entities to freely edit the blockchain [AMVA17, ASL19, AC22]. On the other hand, voting-based approaches establish consensus about warranted redactions in lengthy votes [DMT19, TBM⁺21, MZ19]. As a result, the former approaches violate the desirable decentralization of permissionless blockchains and the latter approaches can only create slow responses to objectionable content.

In conclusion, permissionless blockchains have to react to unwanted content insertion, but there is no obvious path forward: Prevention schemes promise a good deployability to established blockchain systems because they are less invasive, but they cannot guarantee that content never enters the blockchain. Deploying redaction schemes thus becomes inevitable, but they are not suitable for retrofitting established systems, which could leave prevention as the only viable means in this case. We further argue that both approaches can, and should, coexist as prevention strategies can reduce the number of required redactions. In the remainder of this chapter, we thus investigate both approaches to improve the protection for permissionless blockchains against unwanted content insertion.

4.1.1 Constraints for Mitigation Schemes

In this section, we discuss further constraints to potential designs addressing the need for preventing unwanted content from entering a blockchain or establishing means to remove such content after the fact. First, we argue that a full mitigation of content insertion is not possible (Section 4.1.1.1). Thereafter, we discuss requirements for mitigation schemes to serve as guidelines for our designs (Section 4.1.1.2).

4.1.1.1 Impossibility of Full Prevention

We first discuss the most fundamental restriction to any scheme for mitigating blockchain content insertion: *Fully preventing content from being stored on a Bitcoin-like blockchain is not possible.* In the following, we elaborate how Bitcoin’s pseudonymity (cf. Section 2.1) allows (in particular, powerful) content inserters to store their content on the blockchain regardless of the prevention scheme.

Bitcoin’s pseudonymity stems from its handling of coin ownership (cf. Section 2.3.2), which ultimately enables inserting arbitrary data via address manipulation (cf. Section 3.3.1): Every user creates their public-private key pairs locally and shares the derived Bitcoin addresses out-of-band before receiving payments, i.e., without involving the Bitcoin network. As a result, full nodes have no means to validate in advance whether any given (claimed) on-chain address is sensible. For example, they cannot check whether anyone knows the private key corresponding to a P2PKH output. Instead, the full nodes accept any on-chain address in a transaction output in good faith that the output will ultimately be unlocked using the corresponding credentials. Hence, content inserters can store their data in the place of on-chain addresses due to a lack of oversight by the full nodes.

As a hypothetical, assume now that full nodes could ensure that only spendable outputs get accepted into the blockchain. Considering the P2PKH example, this would mean that full nodes could verify that any output’s public-key hash was correctly derived from a corresponding private key. The pre-image resistance of HASH160, which is used to derive the public-key hash (cf. Section 2.3.2.2), makes it computationally infeasible for the content inserter to compute the private key corresponding to any given public-key hash. As a result, the content inserter would not be able to freely choose the (alleged) public-key hash anymore. Additionally, *brute-forcing* manipulated public-key hashes, i.e., randomly choosing private keys until the derived public-key hash coincides with a specific chunk of data, is infeasible as well since HASH160 would not be pre-image resistant otherwise.

Unfortunately, content inserters can significantly lower the difficulty of their brute-forcing approach. By only using parts of the public-key hash for storing content, e.g., the first five of twenty bytes, and by packing transactions with large numbers of outputs of this kind, the content inserter can still encode their content (albeit at a reduced capacity). The resulting transaction consists of only spendable outputs, and thus remains indistinguishable from other Bitcoin transactions in terms of spendability. In fact, vanity addresses are created using this partial brute-force approach as well (cf. Section 2.3.2.2). Even though the maximum capacity would be

reduced by 75 % in this example, content insertion remains computationally possible this way. Hence, content can still be added to entirely valid Bitcoin transactions in a computationally feasible manner, even in the strictest possible scenario.

In conclusion, blockchain content insertion cannot be prevented entirely, and the procedure can only be made harder or more infeasible for content inserters.

4.1.1.2 High-Level Requirements

In this section, we define high-level requirements for approaches to mitigating blockchain content insertion. While these requirements provide general guidelines, we will detail throughout this chapter how they affect prevention and removal differently.

Accuracy. Mitigation schemes must strive for high accuracy levels, i.e., the schemes must allow for detecting as much unwanted content as possible and be able to act accordingly. As we have discussed in Section 4.1.1.1, full prevention of content insertion is infeasible. Thus, inaccurate detection of content-holding transactions may lead to false negatives or false positives, i.e., undetected content-holding transactions or falsely flagged benign transactions. False negatives may lead to serious consequences for full node operators (cf. Section 3.4.2.3). Contrarily, false positives may lead to barring legitimate payments and impede the system’s intended operation. Such negative impacts introduced by mitigation schemes must be kept minimal.

Decentralization. Decentralization is one of the fundamental characteristics of permissionless blockchains. Any mitigation scheme must maintain this decentralization and avoid introducing especially trusted third parties.

Transparency. Mitigation schemes must ensure that participants can transparently assess content-related decisions. In the context of mitigating blockchain content insertion, transparency is achieved if any interested observer can determine from publicly accessible information, such as the blockchain or the consensus rules, why any particular transaction has been determined to hold unwanted content.

Fast Reaction. Once content is being detected in pending transactions or on the blockchain, the full nodes must be able to react quickly to delete unwanted content. Furthermore, proposed mitigation schemes must remain scalable to large numbers of full nodes and transactions so that the fast reaction is not impeded.

Acceptability of Changes. Designs for mitigation schemes must keep deployability in mind. Any update to the consensus rules has the potential to spark controversies, e.g., the discussions about Bitcoin’s maximum block size [Bit15c, Mor17]. Hence, proposed changes of the consensus rules should be minimal or transfer established practices from other systems to increase the changes’ anticipated acceptance. Furthermore, mitigation schemes should respect that limited channels for inserting small chunks of non-financial content are intended even in Bitcoin (cf. Section 3.3.1) to fuel blockchain-backed applications and therefore constitute accepted practices.

Next, we state the contributions of this dissertation toward mitigating unwanted content insertion in Bitcoin-like blockchains within the boundaries set by our analysis of constraints for respective designs.

4.1.2 Contributions

In this chapter, we make two contributions to improve the mitigation of unwanted content with the goal of protecting permissionless blockchains. Our contributions cover both the prevention of content insertion and means to retrospectively remove already persisted content in a coordinated manner:

First, we investigate the design space for *prevention strategies* aiming at retrofitting Bitcoin-like blockchain systems with means to fend off unwanted content before it enters the blockchain (Section 4.2). Namely, we propose three orthogonal prevention strategies and discuss their individual advantages and disadvantages: (a) actively *scanning* for content in pending transactions, (b) establishing *mandatory minimum fees* to financially disincentivize large transactions that are more likely to hold content, and (c) protecting current on-chain addresses by *cryptographically hardening* them against manipulation. Our results show that especially mandatory minimum fees and hardened on-chain addresses could be deployed to Bitcoin with simple changes and reduce content insertion, albeit without preventing it entirely.

Second, we thus present *RedactChain*, a redactable blockchain that enables swift redactions without compromising established trust assumptions by recentralizing control over the blockchain. RedactChain achieves this desirable combination by outsourcing redactions to multiple small redaction juries that are periodically elected at random and that mutually oversee each other’s activities. The redaction juries can only perform redactions via threshold cryptography, i.e., they have to cooperate to alter the blockchain; by keeping the juries small, we reduce performance overheads and ensure a fast reaction. The mutual oversight among multiple redaction juries ensures that single rogue juries cannot make unwarranted redactions nor stall the execution of warranted redactions. Finally, RedactChain enforces moderation-related validation rules to ensure transparency, as juries must log their activities on a dedicated redaction log. This redaction log establishes a consensus about accepted modifications, and thus further limits the potential for harm due to rogue juries.

With these contributions, we highlight the potential for alleviating, but not fully eliminating, the risks associated with unwanted content insertion even for long-running blockchain systems. Moreover, we provide a strategy for designing permissionless blockchains that can react swiftly to content that has entered the blockchain without compromising decentralization, trust assumptions, or transparency.

4.2 Prevention Strategies Against Unwanted Content

As one of the main results of our analyses in Chapter 3, we identified that operators of Bitcoin full nodes may potentially face serious negative consequences if a malicious actor (**P1**, cf. Section 1.1.2) would store objectionable or even illegal content on the blockchain (cf. Section 3.4.2.3). While our in-depth investigation of content on Bitcoin-like blockchains did not confirm the presence of clearly illegal content, we observed different instances of problematic behavior, such as doxing or the insertion

of questionable content (cf. Section 3.5). In one case, content added to the blockchain of Bitcoin SV even triggered police investigations in the UK (cf. Section 3.2.1).

In such cases, it would be desirable or even legally required (e.g., in case of GDPR violations) that the full nodes can delete identified content. However, this approach would violate the blockchain’s immutability property, which helps prevent double-spending attacks (cf. Section 2.1). Namely, the Proof of Work (PoW) puzzle as part of the consensus rules of Bitcoin-like blockchains is designed to explicitly prevent retrospectively modifying the blockchain (cf. Section 2.4.1) and enforce immutability. Any relaxation of the immutability property requires massive coordination among the full nodes and miners (**P3**). Full nodes can erase content they identify and deem objectionable from their local blockchain copy only, but they then depend on other full nodes validating locally erased transactions on their behalf [FHBS19].

An intuitive approach to avoiding these impairments to immutability and public verifiability is to *prevent problematic content from entering the blockchain* in the first place (**Q1**, cf. Section 1.2). In this section, we thus explore *prevention strategies* that can be deployed to Bitcoin-like blockchains with only minimal changes to the consensus rules (**Q2**), i.e., network-wide updates are reduced to single, isolated components, respectively (**P3**). While full mitigation of content insertion cannot be achieved (cf. Section 4.1.1.1), we investigate to which extent simple prevention strategies can provide a heuristic line of defense against the insertion of objectionable content as discussed in Chapter 3.

We explore the resulting design space for prevention strategies from three perspectives. First, we present a *naïve content-filtering approach* and discuss its shortcomings regarding its required network-wide coordination of what content to filter out. Afterward, we investigate possibilities to disincentivize large-scale content insertion by extending Bitcoin’s fee model with *mandatory minimum fees*. Finally, we propose *hardened on-chain addresses* as a replacement for on-chain addresses that enable full nodes to detect simple modifications and require content inserters to now resort to brute-forcing addresses as we described in Section 4.1.1.1.

The results of our analyses are two-fold. On the one hand, we find that the naïve approach of targeted content filtering constitutes a first ad-hoc solution against objectionable content, but it is easily evadable and requires much coordination among the nodes (**P3**). On the other hand, simple adaptations, such as introducing mandatory minimum fees or replacing manipulable on-chain addresses, can effectively deter the insertion of larger blockchain content with moderate overheads (**P2**). Overall, even though prevention strategies can never fully prohibit unwanted content insertion, they can successfully disincentivize the insertion of (especially larger) content.

In the remainder of this section, we first discuss related work (Section 4.2.1) before outlining our approach and the requirements for our prevention strategies (Section 4.2.2). We then present the proposed prevention strategies, i.e., the content-filtering approach (Section 4.2.3), mandatory minimum fees (Section 4.2.4), and hardened on-chain addresses (Section 4.2.5). In those sections, we also assess the individual strengths and weaknesses of the respective strategies. Finally, we conclude with a summary of our results and future work (Section 4.2.6).

4.2.1 Related Work

We now discuss prior work related to mitigating unwanted content insertion that does not weaken the immutability of permissionless blockchains. We first comment on more general approaches to *systems monitoring* before discussing blockchain-specific solutions. Blockchain-specific proposals include *hardening transactions* against simple content insertion, enabling full nodes to *locally erase* content, and *alternative blockchain designs* that are less susceptible to content insertion.

Monitoring Systems. Monitoring incoming data as well as recognizing and filtering unwanted content is a classical application of firewalls, intrusion detection systems, and spam filters [Roe99, IKBS00, HHAZ14, HSS01], which have drastically improved security and quality of service within their respective domains. However, their often high adaptability requirements are commonly tackled via supervised or automated learning of what content should be filtered. The decentralization of permissionless blockchain systems limits the utility of these approaches, as all full nodes have to learn deterministically to take the same decisions. Moreover, all overhead stemming from these local checks multiply over the whole network and, thus, computation-intensive learning processes and checks should be avoided when scanning for unwanted content. To aid the monitoring of blockchains for illegal content, Cremona et al. [CTDR19] proposed to establish a decentralized lookup table to help full nodes identify known instances of illegal content and reject corresponding transactions. This lookup table would contain perceptual hash values of illegal content such as child abuse imagery, similarly to Microsoft’s PhotoDNA¹ service. This extension enables full nodes to reject transactions holding objectionable content, but the lookup table has to be maintained via an additional permissioned blockchain, which potentially impedes the scheme’s decentralization and retrospective deployability.

Hardening Transactions. Beyond attempting to detect and reject pending transactions holding unwanted content, another approach proposed to alter transactions such that encoded content cannot be obtained from the data recorded on the blockchain. In 2013, Gregory Maxwell proposed a hardening scheme for P2SH transaction outputs on the mailing list for Bitcoin development [Max13b]: Instead of recording the script address of a P2SH redeem script in the transaction output directly (cf. Section 2.3.3.3), transaction creators would include only the hash value of the script address to be recorded on the blockchain. Additionally, the transaction creator would have to submit the script address together with its hash value to prove to the network that the transaction creator knows the hash value’s pre-image. Hence, the hash value recorded on the blockchain cannot be chosen arbitrarily anymore. Even if the original script address was encoding a chunk of non-financial data, it would be hard for full nodes, miners, and other Bitcoin users to obtain the manipulated script address based on information they can retrieve from the blockchain. However, this approach requires an additional channel to exchange the raw script addresses (comparable to how the subsequently deployed SegWit operates, cf. Section 2.3.4) and the approach does not address all forms of content insertion, such as input stuffing (cf. Section 3.3.1).

¹<https://www.microsoft.com/en-us/photodna>

Local Erasure. To evade liabilities stemming from storing objectionable blockchain content, full node operators can selectively delete affected transactions from their local blockchain copy. However, simply deleting that data breaks the integrity of the local blockchain copy and the full node cannot validate new transactions referencing deleted older transactions anymore. To remedy this situation, Florian et al. [FHBS19] proposed a functionality-preserving local erasure (FPLE) scheme for Bitcoin-like blockchains, which enables full node operators to remove objectionable content from their local blockchain copy securely, i.e., with minimal impact on the node’s capability to validate new transactions, while keeping compatibility to other nodes in the network a priority. As such, FPLE focuses on cases where personal preferences or differences between jurisdictions cause only a subset of users to erase any given content [FHBS19]. Full nodes using FPLE maintain a local erasure database, which allows them to flag individual (content-holding) transaction outputs for erasure. The full node then refuses to relay any pending transaction referencing a locally erased output, but the full node accepts blocks incorporating such transactions. Hence, the full node relies on other full nodes that did not erase the same affected outputs to validate the transaction properly. Depending on the share of full nodes erasing any given output, FPLE creates the risk of creating forks or enabling the double-spending of the affected outputs [MHMW24]. Ultimately, FPLE remains more stable if no majority of nodes locally erases the same outputs [MHMW24].

Alternative Blockchain Designs. Content insertion does not affect all blockchain designs in the same way. While Bitcoin-like systems are prone to the insertion methods and the consequences we discussed in Chapter 3, alternative designs have emerged that are more resilient against unwanted content insertion. Most notably, balance-based blockchain systems for managing cryptocurrencies [Bru14, CLO16, Poe16, SM19] promise to naturally prevent content from being inserted. These cryptocurrencies do not simply append accepted transactions to the transaction ledger, but the transactions’ effects are aggregated directly in a more succinct state. For instance, the mini blockchain scheme [Bru14] manages coin ownership by maintaining accounts and associated balances instead of a UTXO set. Even though the primary focus of this aggregation is reducing the required blockchain history (we extensively consider this aspect in Chapter 5), persisting content also becomes hard in these aggregated structures. For instance, the mini blockchain scheme enables this aggregation of balances by completely removing Bitcoin’s scripting system [Bru14]. However, these designs strongly deviate from Bitcoin’s initial design and thus are not easily deployable retrospectively.

In conclusion, several prior proposals can help prevent blockchain content insertion, either directly or indirectly. However, these approaches are either infeasible to deploy or pose challenges for the coordination of full nodes.

4.2.2 Problem Statement

Investigating suitable strategies that allow full nodes to prevent content from entering the blockchain promises the fastest possible means to mitigate negative consequences of the insertion of objectionable content: Instead of having to react, the full

nodes could proactively keep unwanted content off the blockchain. Hence, the main goal of our first contribution toward mitigating blockchain content is to explore and assess available prevention strategies.

We have identified in Chapter 3 that Bitcoin-like blockchains already contain questionable content (cf. Section 3.5.3.1) and that such content can easily be added to those blockchains using unintended ways such as address manipulation (cf. Section 3.3). As a result, Bitcoin-like blockchains are in need of immediate attention. We thus focus on hardening Bitcoin-like blockchains against unwanted content insertion in the following.

However, as we have discussed in Section 4.1.1.2, mitigation schemes have to satisfy more requirements beyond enabling a fast reaction. Our prevention strategies therefore need to be developed with deployability to Bitcoin and related systems in mind. Hence, we focus our efforts on *simple* prevention strategies that only require minor changes to Bitcoin’s consensus rules. This way, we can ensure that our prevention strategies maintain decentralization and transparency (cf. Section 4.1.1.2).

Since content insertion cannot be fully prevented (cf. Section 4.1.1.1), we have to assess the accuracy of our proposed prevention strategies, i.e., the probability of generating false positives or false negatives. Moreover, not all content insertion is deemed unwanted. In Section 3.4.1, we have highlighted the multitude of applications enabled by the capability of pegging short chunks of non-financial data to Bitcoin’s blockchain. We observe that restricting content insertion to such *short* data chunks is beneficial to enable those applications, and tighter size limits make it increasingly harder for content inserters to encode objectionable content in these short chunks. Even short links to objectionable content do not put full node operators at risk of possessing said content: As the content is not stored directly on the blockchain, operators do not own a physical copy of it. Contrarily, *arbitrary-length* data chunks allow for encoding complete image files and other objectionable or even illegal content. Unfortunately, content insertion methods applicable to Bitcoin-like blockchains allow for encoding large files in ways that they are easily recoverable again, e.g., via content insertion services (cf. Section 3.3.2). Prevention strategies should be developed with this asymmetry in mind and allow the insertion of short data chunks while discouraging the insertion of arbitrary-length content. For the scope of our discussions, we consider short-sized content that can be encoded using at most 1 KiB to have a lower harm potential than arbitrary-length content. Furthermore, we consider content that can be encoded using at most 100 Byte harmless, as these especially short data chunks fuel a plethora of benign blockchain applications and Bitcoin actively offers `OP_RETURN` as an intended way to insert up to 80 Byte per transaction (cf. Section 2.3.3.3). These thresholds may, however, require further adaption in the future, e.g., due to future court rulings.

Summary of Problem Statement. The goal of our contribution is to explore the design space for *prevention strategies* that (a) prevent harmful content from entering the blockchain, (b) are easily deployable to Bitcoin-like blockchains, and (c) are adaptable in case that also short-sized blockchain content reveals currently unforeseen risks. To this end, the prevention strategies shall (a) *allow* very short content (≤ 100 Byte), (b) *tolerate* medium-sized content (≤ 1 KiB), and (c) effectively *prevent*

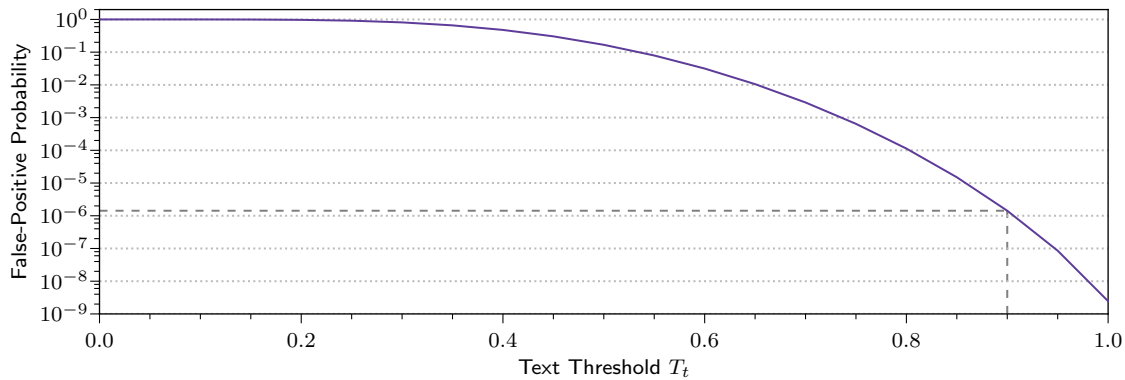


Figure 4.1 The expected false-positive rate when attempting to detect text-based content in a 20 Byte-long on-chain address decreases for larger text thresholds T_t , but only large thresholds yield acceptable results.

the insertion of arbitrary-sized content. These thresholds need to remain adaptable if deemed necessary, e.g., by court rulings. Furthermore, as a complete prevention of content insertion is impossible, our the prevention strategies shall render the insertion of arbitrarily sized content either computationally or financially infeasible. In the following, we discuss the investigated strategies' applicability and effectiveness directly after presenting them. We base our discussion on the high-level requirements for content mitigation schemes we presented in Section 4.1.1.2, i.e., we judge to which extent the prevention strategies are accurate, maintain the decentralization of the underlying blockchain system, provide transparency, allow for fast reaction to content, and we assess the deployability of the proposed strategies.

4.2.3 Filtering Content-Holding Transactions

We first discuss an intuitive strategy to prevent blockchain content insertion and identify shortcomings of this naïve approach: In Section 3.5.1, we presented our framework of *content detectors* we used to scan Bitcoin-like blockchains for content. Intuitively, these content detectors should also be applicable to detect and reject any pending transaction that makes use of unintended content insertion methods. This approach would extend permissionless blockchain systems with a functionality similar to the deployment of current antivirus software, as the full nodes would obtain a list of rules according to which they identify objectionable transactions.

In the following, we consider two examples for content detectors as a basis for our further discussions. Afterward, we discuss consequences regarding the deployability of using content detectors for filtering pending transactions. Both considered content detectors aim at identifying immediately meaningful content inserted via address manipulation (cf. Section 3.3.1). First, we consider an adapted *text detector*, which has the goal of identifying transactions carrying human-readable texts or text-based files. Second, we consider a hypothetical *known-file detector*, which scans transactions for binary files, such as images or archives.

Text Detector. Detecting large fractions of printable text within a transaction can be used to prevent custom text and text-based files, such as HTML pages or Python

scripts, from being stored on the blockchain. In Section 3.5.1.2, we presented a content detector aimed explicitly at scanning for text-holding transactions. For our discussions, we consider a slight variant of that content detector: Instead of looking for consecutive printable ASCII characters, we now define a *text threshold* $T_t \in [0, 1]$ and check whether the fraction f_t of printable ASCII characters in a transaction output's on-chain address exceeds that threshold, i.e., we check whether $f_t \geq T_t$. Choosing T_t appropriately depends on the probability that random on-chain addresses resemble printable text by chance. Figure 4.1 shows the probability that the text detector flags a randomly chosen on-chain address from a P2PKH output (i.e., a 20 Byte-long public-key hash) for containing a share of $f_t \geq T_t$ printable ASCII characters for increasing text thresholds T_t (assuming 95 out of 256 characters are printable). This probability yields an expected false-positive rate (FPR) for falsely flagging benign transaction outputs. We observe that only high text thresholds lead to negligible FPRs: While $T_t = 0.75$ still yields an expected FPR of 0.064%, $T_t = 0.9$ yields an expected FPR of 0.000142%. We thus suggest choosing $T_t \geq 0.9$. Unfortunately, even small expected FPRs can have a strong impact on benign payments. Some users choose to reuse a Bitcoin address, e.g., they may publish a single address to collect donations. For instance, we identified a Bitcoin address that would be flagged by the text detector but has received bitcoins in 360 transactions² and thus constitutes a false positive. One solution to this problem is to consider transactions as content-holding only if the text detector considers multiple outputs content-holding. For instance, introducing an *output-count threshold* T_c allows flagging transactions only once $f_t \geq T_t$ holds for at least $n \geq T_c$ distinct transaction outputs. Choosing $T_c = 5$ would thus permit inserting of up to 100 Byte of data per transaction, i.e., our threshold for considering data chunks harmless as defined in Section 4.2.2.

Known-File Detector. After having outlined the challenges of detecting text-based content in Bitcoin transactions as well as potential remedies, we now focus on detecting binary files, such as images or compressed archives. In contrast to text-based content, we deem the simple detection of binary files infeasible using a comparable approach: Instead of checking for large shares of printable ASCII characters, a hypothetical known-file detector has to consider other notable structures to identify binary files embedded into transactions. Binary files are typically identifiable via *magic numbers*, i.e., short byte sequences that are unique to the respective file type; typically, however, these sequences are often only a few bytes long [Kes22]. Hence, the expected FPR increases drastically as the probability increases that a random byte sequence coincides with a file type's magic number. Although the accuracy can be improved by considering more characteristic features, e.g., whole file headers instead of only relying on magic numbers, content inserters can evade overly specific detectors more easily. For instance, a content inserter can introduce trivially reversible modifications such as padding the file with a deterministic number of random bytes to prevent a highly specialized content detector from flagging the transaction.

Deployment Considerations. Full nodes need to be able to update their local set of applied content detectors to remain capable of reaching consensus with the other nodes. Hence, the network would have to implement means to (a) agree on a set of

²Bitcoin address: 154QWLN3Uz43nHMAM7ioYUx8tkYXdNKDtQ, last bitcoins received on Jul 9, 2016

content detectors to apply and (b) distribute these content detectors to the nodes. The content detectors could, for instance, be fixed and distributed by the full node developers via updates to the client; however, this approach arguably reduces the degree of decentralization in the network if content-filtering policies become increasingly complex and if they are not necessarily following a prior discussion. Alternatively, the blockchain system could be extended with a new component for negotiating and distributing new content detectors. Unfortunately, this approach is infeasible for assessing pending transactions, as the introduction of new content detectors requires thorough review. As a result, the deployment of detector-based content prevention faces further open challenges.

Discussion

Actively scanning for content-holding transactions using content detectors is an intuitive strategy to prevent unwanted content insertion on permissionless blockchains. We have discussed that, in principle, content detectors can be fine-tuned to reject transactions with embedded unwanted content with high accuracy, i.e., we can reduce the number of expected false positives to protect honest users from being affected by our changes. However, the example of the known-file detector illustrates that the filtering quality of general-purpose content detectors can also be limited and may lead to frequent updates of the set of applied detectors.

Moreover, content inserters would inevitably have access to the deployed content detectors, since we are considering permissionless blockchain systems. Hence, malicious content inserters can develop strategies to evade the content detectors, as illustrated by the possibility to pad binary files we discussed before. Honest full node operators have to react to such evasion strategies by negotiating, distributing, and applying an increasing number of content detectors via *mandatory* updates. Otherwise, the full nodes would not be able to reach consensus anymore about which transactions to accept or reject. The expected frequency of mandatory updates of the applied content detectors can ultimately lead to uncontrolled forking and drastically decreases the deployability of this prevention strategy. Hence, we argue that maintaining transparency is a necessary precondition for deploying content detector-based filtering, but the required coordination increasingly impedes the network's ability to react fast to the discovery of new content insertion methods as well as the expected acceptability of the (frequent) changes to the deployed content detectors.

In conclusion, explicit content detection constitutes a first line of defense against unwanted blockchain content and can be fine-tuned to work for text-based content. Mitigating the insertion of unwanted binary files, however, remains challenging when relying on content detectors alone. Thus, we investigate further, implicit strategies for preventing content insertion in the following.

4.2.4 Mandatory Minimum Transaction Fees

The alternative to actively scanning for unwanted content in pending transactions is to deter its insertion indirectly. In this section, we investigate the applicability of

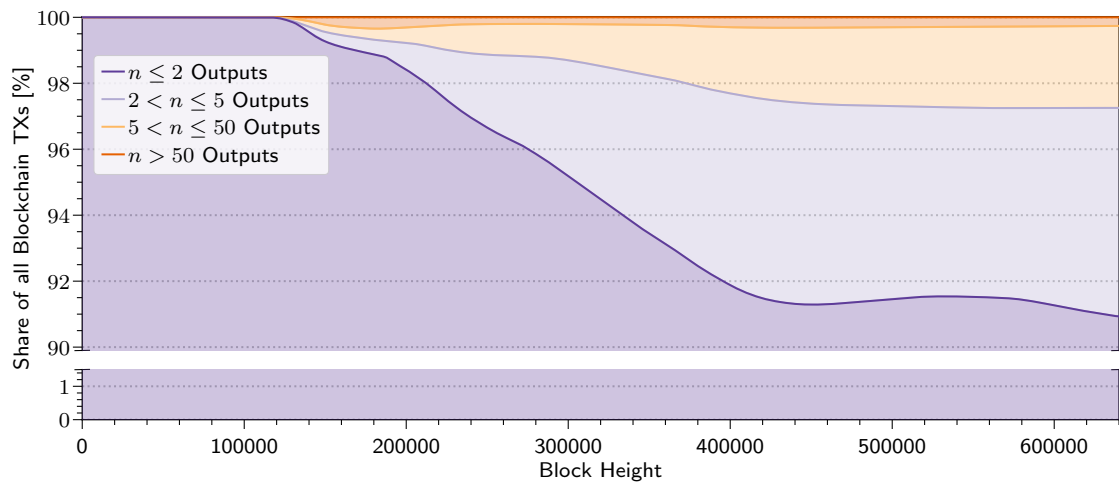


Figure 4.2 The evolution of shares of tiny, small, medium, and large transactions stored on Bitcoin’s blockchain as new blocks are added indicates that the share of medium-sized and large transactions increased over time. However, small transactions with at most five outputs still constitute over 97 % of all transactions at block height 640 000 (from Jul 20, 2020).

mandatory minimum fees to financially disincentivize creating unnecessarily large transactions, which could, in turn, be used to insert content.

This prevention strategy exploits the fact that Bitcoin-like systems feature transaction fees. These fees are voluntary, but paying more fees creates a stronger incentive for miners to include a transaction in their block [Pri20]. As we outlined in Section 2.4.1, the fees are expressed in relation to a transaction’s size. Typically, transaction fees are given in Sat/Byte; Bitcoin Core gives the fees in Sat/WU since the deployment of SegWit (cf. Section 2.3.4). For the sake of generalizability to other systems than Bitcoin Core, we focus on byte-wise transaction fees in the following.

It follows that content inserters have to expect to pay transaction fees in relation to the size of their content to be inserted in addition to any costs of potentially burning coins, e.g., when relying on address manipulation (cf. Section 3.3.1). In times of larger contention due to large numbers of pending transactions, the per-byte fees required to get a transaction stored on the blockchain may increase significantly [Pri20]. As a result, content inserters face an increasingly relevant financial barrier, especially when they attempt to insert large amounts of data. However, expected transaction fees fluctuate considerably over time. While we experienced a first peak in expected transaction fees in Q4 of 2017, where transaction creators had to pay fees of 269 Sat/Byte on average, the average expected fees were only 4 Sat/Byte in Q3 of 2022 [Pri20]. These fluctuations heavily impact the cost of content insertion, i.e., high expected fees make content insertion more expensive and low expected fees do not deter the practice on a financial level.

This observation motivates our second prevention strategy: Instead of only creating incentives to include transactions in a block via voluntary fees, miners and full nodes can actively disincentivize the submission of content-embedding transactions by enforcing the payment of *mandatory minimum fees*. In addition to letting miners prioritize transactions paying higher fees, all full nodes would then reject transac-

tions that underpay these mandatory fees. By scaling these mandatory minimum fees with the number of transaction outputs, the network can penalize larger transactions, which are more suitable for embedding illicit content, stronger and limit the negative impacts on smaller transactions. Figure 4.2 shows how the cumulative share of tiny (at most two outputs), small, medium-sized, and large transactions stored on the blockchain of Bitcoin Core evolved over time up to the block height 640 000, which we considered for our blockchain analyses in Section 3.5.2. Over time, medium-sized and large transactions slightly increased their share, i.e., they became more frequent on Bitcoin’s blockchain. However, tiny and small transactions together still constitute 97.3% of all transactions on Bitcoin’s blockchain, whereas large transactions only make up 0.3% of all transactions. Hence, our approach of scaling mandatory fees with the number of transaction outputs has the potential to disincentivize the creation of very large, and uncommon, transactions while avoiding negative impacts for the vast majority of Bitcoin transactions.

To further illustrate our proposed adaptations to Bitcoin’s current voluntary fee model, we model the transaction fees to be paid by a transaction creator as:

$$F_v(t) = \alpha \cdot s_t$$

Here, s_t is the transaction’s size in byte and α is the expected per-byte transaction fee to likely get the transaction included within the next six blocks (or roughly an hour) [Pri20]. In this model, the transaction creator can choose to pay lower fees than $F_v(t)$ at the risk that their transaction is further delayed or omitted completely. Instead of letting miners handle transaction fees based on their financial incentives, we propose to introduce a new validation rule to be enforced by the full nodes that each transaction pays a *mandatory* minimum fee $F_m(t)$.

By further adapting how these mandatory minimum fees are computed, we can penalize the creation of large transactions by scaling the required fees with a transaction’s number of outputs n_t . Initially, we proposed to virtually increase the considered transaction size depending on the number of outputs [MHZ⁺18] by setting

$$F_m^\dagger(t) = \alpha \cdot (s_t + \beta(n_t) \cdot n_t)$$

where $\beta(n_t)$ is a scaling function that serves to increasingly penalize transactions as their number of outputs grows. Even though fitting the situation of Bitcoin in late 2017 [MHZ⁺18], this approach has two potential shortcomings given the evolution of transaction fees: First, the mandatory minimum fees directly scale with the expected per-byte fees α . While this approach helps to increase the penalties for content insertion when fees are high, low per-byte fees may significantly attenuate the additional cost of content insertion. Second, the penalty itself is determined based on the number of transaction outputs n_t , which is generally considerably lower than the overall transaction size s_t ; for instance, one P2PKH transaction output has a size of 25 Byte. Hence, the penalty for content insertion does not grow as fast as the expected transaction fees, which are based on the total transaction size.

We thus propose a simple revision as an alternative to our initial model for mandatory minimum fees as follows:

$$F_m^*(t) = (\alpha + \beta(n_t)) \cdot s_t$$

This adaption essentially makes the penalty a true secondary per-byte transaction fee, while still allowing to scale the penalty based on the number of transaction outputs. Hence, $F_m^*(t)$ counteracts both problems from before by implementing a fixed base penalty that is independent of the expected per-byte fee α and by applying the penalty on a per-byte basis instead of a per-output basis.

The strength of the applied penalties is determined by the choice of $\beta(n_t)$. We propose a piecewise definition for $\beta(n_t)$ to account for the different classes of transactions we discussed in Section 4.2.2. To recapitulate, we consider transactions embedding up to 100 Byte harmless, transactions embedding up to 1 KiB tolerable, and all larger transactions potentially harmful, i.e., capable of embedding unwanted content. Considering P2PKH-based address manipulation, we can directly map these requirements to corresponding thresholds of $T_s = 5$ and $T_m = 50$ transaction outputs, as each P2PKH output can hold 20 Byte of data (cf. Section 3.3.1). Using these thresholds, we can assign a constant penalty strength to each transaction class:

$$\beta_C(n) = \begin{cases} 0 & n \leq T_s \\ P_m & T_s < n \leq T_m \\ P_l & n > T_m \end{cases}$$

Here, P_m and P_l are the tunable penalties applied to medium-sized and large transactions, respectively. We assign no penalty to small transactions so that the vast majority of benign financial transactions remains unaffected by our prevention strategy targeting unwanted content insertion. We can further aggravate the penalties for increasingly large transactions, e.g., by considering a piecewise linear penalty instead of a piecewise constant penalty:

$$\beta_L(n) = \sum_{i=1}^n \beta_C(n)$$

These two examples for penalty functions serve to provide an intuition about how full nodes can financially penalize increasingly large transactions, which are suitable to hold content and statistically uncommon to occur naturally as financial transactions. Moreover, large financial transactions can be split into smaller or medium-sized transactions more easily than content-holding transactions. In the following, we discuss examples of effective penalties that can be enforced using these simple adaptations to Bitcoin's fee model.

Discussion

We now assess the potential of mandatory minimum fees to deter the insertion of content in Bitcoin-like blockchains. To this end, we first evaluate the mandatory fees our example penalty functions $\beta_C(n)$ and $\beta_L(n)$ would inflict in different scenarios; however, our discussions focus on general properties of the generated penalties, and we deliberately leave the exact determination of penalty functions and their parameters for future work. Afterward, we discuss to which extent mandatory minimum fees can satisfy the high-level requirements we put forward in Section 4.1.1.2.

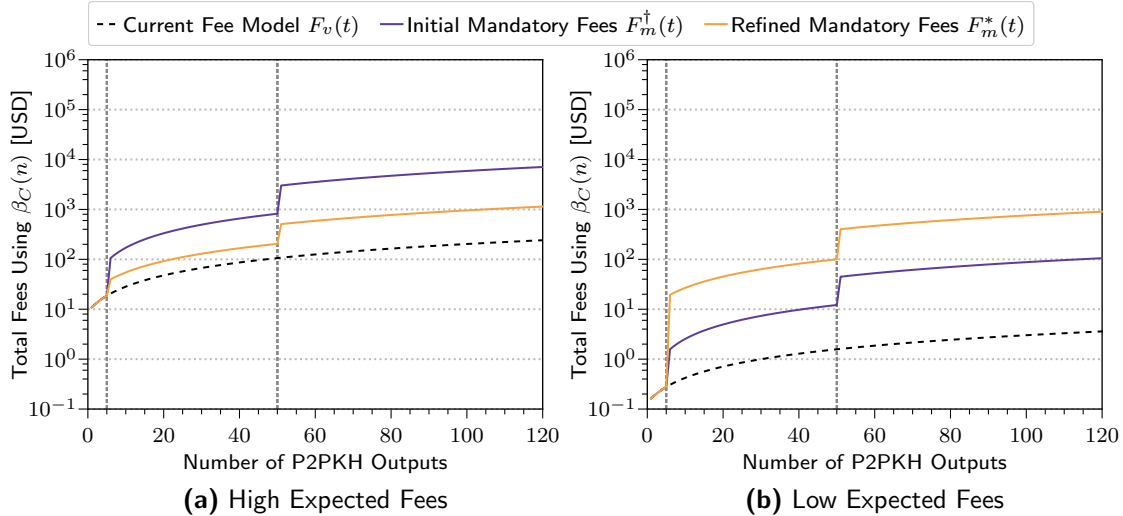


Figure 4.3 A piecewise constant per-output penalty $\beta_C(n)$ allows elevating the mandatory minimum fees according to the classes of transaction sizes, i.e., small transactions are not subject to any penalty, medium-sized transactions can be moderately penalized, and large transactions have the strongest penalty assigned. Our refined model $F_m^*(t)$ creates smaller relative penalties compared to our initial proposal $F_m^\dagger(t)$ when the expected per-byte fees α are higher, but $F_m^*(t)$ remains more stable when those expected fees are especially low. However, $\beta_C(n)$ generates only moderate overall penalties that do not increase further for extremely large transactions.

For our subsequent considerations, we assume a Bitcoin user who creates a transaction t that consists of one P2PKH input and an increasing number n of P2PKH transaction outputs, and we assess the mandatory minimum fees that user has to pay for their transaction to be added to the blockchain. We assume a market price of 21 198.80 USD/BTC, which was the average market price in Q3 of 2022 [Blo11b]. Furthermore, we distinguish two scenarios by considering both high and low expected per-byte fees. We set $\alpha_H = 269$ for our high-fee scenario (corresponding to the first peak of expected transaction fees in Q4 of 2017) and $\alpha_L = 4$ for our low-fee scenario (corresponding to the comparably low fees experienced during Q3 of 2022). For each scenario, we also compare the penalties inflicted by our refined fee model $F_m^*(t)$ to those inflicted by our initial fee model $F_m^\dagger(t)$. Finally, we set the strengths of the penalties for creating medium-sized and large transactions to $P_m = 250$ and $P_l = 1000$, respectively. We chose these values as examples so that creating a medium-sized transaction imposes an extra penalty that roughly corresponds to previous high transaction fees (i.e., α_H), and creating a large transaction should result in a considerably stronger penalty than that base penalty; however, the penalty strengths can be adjusted as needed.

Figures 4.3 and 4.4 show the total mandatory minimum fees the Bitcoin user has to pay when adopting a piecewise constant ($\beta_C(n)$, Figure 4.3) and linear ($\beta_L(n)$, Figure 4.4) per-output penalty underlying the fee model, respectively. Both fee models increasingly penalize large transactions by scaling the mandatory minimum fees with the number of transaction outputs. Since $\beta_L(n)$ effectively accumulates the penalties for all transactions with $n' \leq n$ outputs, the penalties inflicted by that

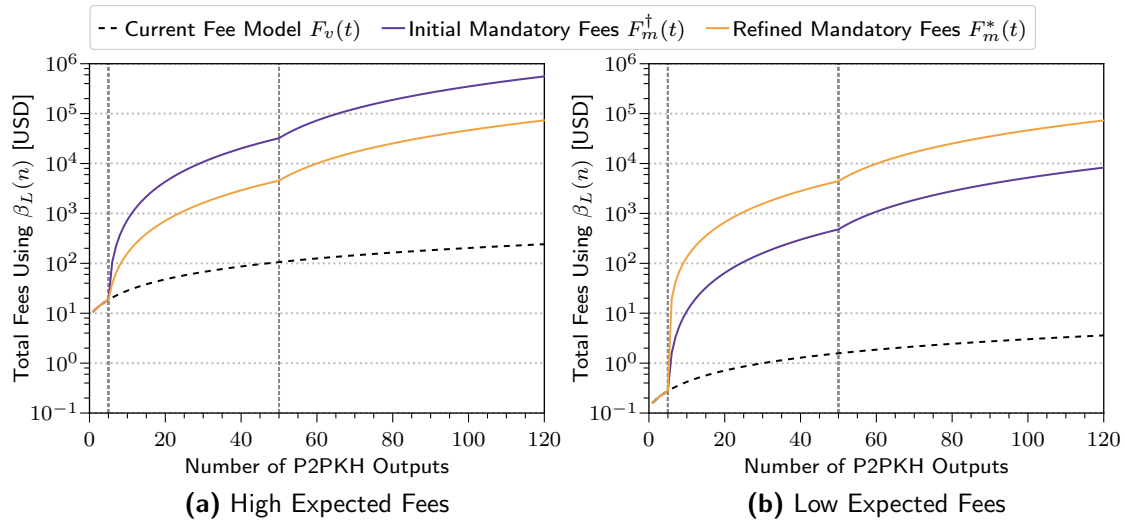


Figure 4.4 Scaling the penalty linearly with the number of transaction outputs via $\beta_L(n)$ maintains the same basic properties between $F_m^*(n)$ and $F_m^\dagger(n)$, but the approach has the additional benefit of scaling the penalty within one class of transaction size. Since every additional transaction output further increases the penalty, the approach creates stronger disincentives for creating exceptionally large transactions when compared to $\beta_C(n)$.

penalty model can become significantly larger than those inflicted when using $\beta_C(n)$. For instance, submitting a transaction with $n = 100$ P2PKH outputs (corresponding to just 2 kB of content) would cost 203 USD in the current fee model when expected per-byte fees are high. Inserting the same transaction with mandatory minimum fees according to our refined fee model $F_m^*(t)$ would cost the user roughly 957 USD using $\beta_C(n)$ as the penalty model (Figure 4.3a) and roughly 46 388 USD when using $\beta_L(n)$ (Figure 4.4a). When the expected per-byte fees are low, the user has to pay only 3 USD in Bitcoin’s current fee model. However, the absolute penalties are independent of α when setting mandatory minimum fees based on $F_m^*(t)$ and the user only has to pay slightly fewer total fees when the expected base fee is α_L instead of α_H (757 USD for $\beta_C(n)$ and 46 188 USD for $\beta_L(n)$). This observation highlights a desirable robustness of our refined model $F_m^*(t)$ compared to our initial proposal $F_m^\dagger(t)$. Even though $F_m^\dagger(t)$ creates stronger penalties than $F_m^*(t)$ considering large transactions when expected transaction fees are high in the current fee model (Figures 4.3a and 4.4a), that penalty gets reduced considerably and falls below the penalties according to $F_m^*(t)$ when fees are low.

In conclusion, mandatory minimum fees can be parametrized to create increasingly strong disincentives to submitting exceptionally large transactions that potentially hold objectionable content. For example, a content inserter who wants to embed 50 kB of data in a transaction using 2500 P2PKH outputs would only have paid about 72 USD in fees (on average) during Q3 of 2022, but the fees would add up to roughly 18 100 USD using $\beta_C(n)$ and even over 44.43 million USD using $\beta_L(n)$ when mandatory minimum fees would be computed according to $F_m^*(t)$. Hence, mandatory minimum fees can indeed create a strong deterrent to content insertion.

Mandatory minimum fees further provide decentralization and transparency, and they enable fast reactions to content insertion because they are realized as a simple additional consensus rule known to all full nodes and miners that is enforced via local computations only. Even though the simplicity of this proposal is beneficial regarding a possible adoption by the Bitcoin network, the exact parametrization is a potential conflict point similarly to previous discussions about the maximum block size [Bit15c, Mor17]. Hence, mandatory minimum fees are easily deployable on a technical level, but the exact instantiation of a penalty model would have to withstand scrutiny from the Bitcoin community.

4.2.5 Hardened On-Chain Addresses

In the previous sections, we have presented approaches to detect content-embedding transactions and financially disincentivize their creation. As our third prevention strategy, we propose to make it computationally hard for content inserters to embed arbitrary content in their transactions. The goal of our prevention strategy is to make the currently easy practice of address manipulation (cf. Section 3.3.1) as hard for content inserters as possible. Namely, instead of simply replacing a manipulable on-chain address with a chunk of arbitrary data, the content inserter should be required to resort to a (partial) brute-force attack to insert content, as we outlined in Section 4.1.1.1.

Overview. To achieve this goal, we replace the current on-chain addresses with *hardened on-chain addresses* that are not easily manipulable to insert content. Intuitively, we still rely on Bitcoin’s current on-chain addresses for validation purposes, but we only store a hardened version of those addresses on the blockchain, such that blockchain observers cannot reconstruct manipulated, i.e., potentially content-holding, on-chain addresses anymore. As such, a hardened on-chain address must be *one-way*, *self-verifying*, and maintain *freshness*:

When a hardened on-chain address $\mathcal{C}(x)$ is *one-way*, it becomes infeasible to derive x , a potentially manipulated on-chain address, from $\mathcal{C}(x)$. Hence, $\mathcal{C}(x)$ prevents that transaction outputs record manipulated and potentially content-embedding on-chain addresses on the blockchain. Contrarily, the validation of payments remains simple, as releasing x in a subsequent transaction’s input would enable full nodes to recompute $\mathcal{C}(x)$ to attest that x matches $\mathcal{C}(x)$.

We also require that $\mathcal{C}(x)$ is *self-verifying*, by which we mean that any manipulation of the hardened on-chain address must be easily detectable by full nodes from $\mathcal{C}(x)$ alone. With this property, we prevent a content inserter from simply replacing $\mathcal{C}(x)$ instead of x with a chunk of content.

Finally, hardened on-chain addresses must also satisfy a *freshness* requirement, i.e., sending two payments to the same on-chain address x must result in two different hardened on-chain addresses $\mathcal{C}(x) \neq \mathcal{C}'(x)$. This property mitigates rainbow attacks where content inserters could repeatedly vary x until $\mathcal{C}(x)$ is suitable to be reused often when encoding content, e.g., when $\mathcal{C}(x)$ constitutes a part of a file header.

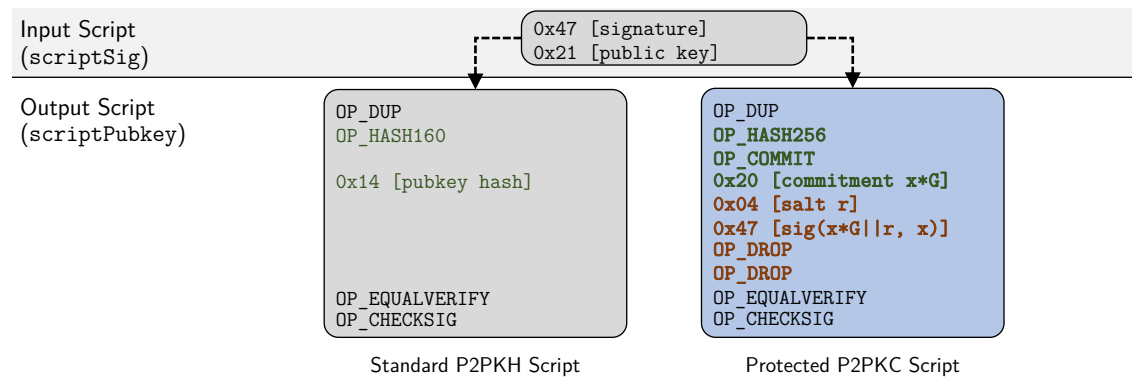


Figure 4.5 Hardened on-chain addresses can be implemented by replacing the manipulable on-chain address x in a standard transaction output with a commitment value $x \cdot G$, a salt r and a signature $\text{sig}(x \cdot G || r, x)$ created by treating x as a private key. Only the commitment value is relevant for validating the authenticity of a payment, i.e., the salt and signature can be dropped from the stack during script evaluation. This adaption yields the Pay to Public-Key Commitment (P2PKC) transaction pattern to replace Bitcoin’s P2PKH pattern; an analogously defined Pay to Script Commitment (P2SC) pattern can replace Bitcoin’s P2SH pattern.

Construction. We construct a hardened on-chain address $\mathcal{C}(x)$ from a regular on-chain address x by deriving a *salted and self-signed cryptographic commitment*:

$$\mathcal{C}(x) := (x \cdot G, r, \text{sig}(x \cdot G || r, x))$$

Here, G is the generator defined for Bitcoin’s elliptic curve `secp256k1` and r is a pseudo-random salt value derived from the transaction identifiers referenced in the current transaction’s inputs. By storing only a commitment $x \cdot G$ instead of x and an ECDSA signature created by interpreting x as an ECDSA private key on the blockchain, we ensure that $\mathcal{C}(x)$ is one-way, as obtaining x from either $x \cdot G$ or the signature is considered computationally infeasible [HMOV04]. However, full nodes can easily recompute $x \cdot G$ and validate $\text{sig}(x \cdot G || r, x)$ when the coin owner submits x in a future transaction input, as r can be obtained from $\mathcal{C}(x)$ and G is publicly known. Further, $\mathcal{C}(x)$ is self-verifying, as manipulating any component of $\mathcal{C}(x)$ would invalidate the signature. Since x is unknown for manipulated $x \cdot G$, a content inserter cannot compute the signature $\text{sig}(x \cdot G || r, x)$ when attempting to store content in $x \cdot G$. Finally, we incorporate the pseudo-random salt r to achieve freshness of $\mathcal{C}(x)$. To ensure with high probability that r cannot be reused, we derive r from the unique context of the transaction containing $\mathcal{C}(x)$. Namely, we compute $r = \text{CRC32}(h(t_1) || \dots || h(t_n))$, where $h(t_i)$ is the transaction identifier of the previous transaction accessed by the i -th input of t . By deriving r via CRC32, we keep the salt small with a size of 4 Byte and avoid unnecessarily bloating the blockchain.

Implementation. We can integrate hardened on-chain addresses into the scripting system of Bitcoin by slightly adjusting its currently accepted standard script patterns (cf. Section 2.3.3.3). This adjustment only requires adding one new `OP_COMMIT` operation to `SCRIPT` that computes $x \cdot G$ from x , as full nodes must be capable of matching both values similarly to recomputing the public-key hash from a submitted public key when validating a P2PKH input-output pair. The `OP_COMMIT` operation

can be added by, for instance, repurposing one of Bitcoin’s reserved opcodes, such as `OP_NOP1` (cf. Section 2.3.3.1).

With this additional operation, we can derive hardened replacements for the most common transaction types, P2PKH and P2SH, which we refer to as *Pay to Public-Key Commitment (P2PKC)* and *Pay to Script Commitment (P2SC)*. Figure 4.5 illustrates the transition from P2PKH to P2PKC. Instead of the manipulable public-key hash x , P2PKC output scripts push the commitment $x \cdot G$, the salt r , and the signature $\text{sig}(x \cdot G || r, x)$, i.e., the three components of $\mathcal{C}(x)$, onto the stack. A P2PKC output script is similar to a P2PKH output and requires the same input script. Instead of deriving the public-key hash using `OP_HASH160`, we compute the commitment $x \cdot G$ by first hashing the public key using `OP_HASH256` and then executing `OP_COMMIT`; we use `OP_HASH256` instead of `OP_HASH160` to better utilize the order of `secp256k1`. The salt and signature are only required for validating the integrity of $x \cdot G$ during the extended checks of pending transactions (cf. Section 2.4.2), but they are not needed for validating whether a subsequent transaction is allowed to access the associated coins. Hence, r and $\text{sig}(x \cdot G || r, x)$ are omitted during the script validation via the `OP_DROP` operation. Afterward, `OP_EQUALVERIFY` continues to check that the recomputed commitment matches the one recorded in the transaction output. If this check passes, the submitted public key is proven eligible to access the coins, and we can check the submitted signature against this key using `OP_CHECKSIG` to verify that the transaction creator knows the corresponding private key. As a result, x is only released to the blockchain if its commitment equals $x \cdot G$ and x can be used to validate the submitted signature, i.e., x is the public key corresponding to the coin owner’s private key. Hence, our P2PKC pattern covers the same spendability condition as P2PKH and additionally hardens the reference to the eligible public key against simple manipulation. P2SC replaces P2SH analogously.

Discussion

In the following, we assess the performance and storage overheads created by hardening on-chain addresses against manipulation, and we discuss to which extent this prevention strategy satisfies the requirements we defined in Section 4.1.1.2.

Performance Overhead. Using hardened instead of manipulable on-chain addresses creates an additional validation overhead for full nodes: On the one hand, the full nodes have to validate $\text{sig}(x \cdot G || r, x)$ in a P2PKC output before accepting a pending transaction. On the other hand, they have to execute `OP_COMMIT` when that transaction output is to be spent after inclusion on the blockchain, i.e., the full nodes have to compute $x \cdot G$ for each accessed P2PKC output. To assess the expected overhead created by these changes, we used `openssl speed` to benchmark ECDSA operations on a commodity PC (Intel Core 2 Quad Q9400 CPU running at 2.66 GHz with 4 GiB RAM) as validating $\text{sig}(x \cdot G || r, x)$ is an ECDSA verification operation, and we approximate recomputing the commitment $x \cdot G$ with an ECDSA signing operation. Verifying a signature takes 0.4 ms and a signing operation takes 0.2 ms in our experiments; hence, accepting a P2PKC output onto the blockchain and a subsequent validation of the script creates a total overhead of roughly 0.6 ms. When

we consider the dataset we used in Chapter 3 starting from 2017, the average Bitcoin Core block contained roughly 1900 transactions. Hence, if we conservatively assume that every transaction had five outputs, hardened addresses would inflict an additional overhead of 3.8s to validate the commitments' signatures of pending transactions and 1.9s to recompute the address commitment when validating an attempted payment, creating a total expected overhead of 5.7s per block. If we assume that every transaction had an exceptionally large number of 50 outputs, hardened addresses would add up to 57s to the validation of each block. As new blocks are only published every 10 min on average (cf. Section 2.4.1), this additional check is clearly feasible. The final changes introduced by hardened addresses are replacing `OP_HASH160` with `OP_HASH256` and deriving the salt using CRC32. Notably, `OP_HASH256` outperforms `OP_HASH160` for small input sizes; furthermore, computing the CRC32 checksum is negligible compared to the other operations. Hence, using hardened addresses instead of manipulable ones does not impede payment validation.

Storage Overhead. Transitioning from manipulable to hardened on-chain addresses increases the size of transaction outputs, as illustrated in Figure 4.5. Namely, we replace the 20 Byte-long on-chain addresses of P2PKH and P2SH transaction outputs with self-verifying commitments of 108 Byte. Here, the commitment $x \cdot G$ has the same form as a compressed ECDSA public key and the signature $\text{sig}(x \cdot G || r, x)$ is also created using ECDSA; hence, when serialized for inclusion in the output script, the commitment has a size of 33 Byte, and we can assume that the signature has a size of 71 Byte (cf. Section 2.3.2.1). Finally, the salt r has a size of 4 Byte when using CRC32. The final P2PKC output script has a total size of 118 Byte compared to the 25 Byte of a P2PKH output script, because P2PKC requires three additional operations (`OP_COMMIT` and `OP_DROP` twice) as well as pushing three values instead of one, resulting in an additional overhead of 5 Byte to adjust the script's logic. To put this increase into perspective, we consider how many P2PKC transactions a single Bitcoin block can hold when using a maximum block size of 1 MB (cf. Section 2.2.2). The majority of Bitcoin transactions has a most two outputs (90.9% of transactions, cf. Figure 4.2). A transaction with one P2PKH input and two P2PKH outputs has an expected size of 225 Byte. A P2PKC transaction of the same format has a size of 411 Byte instead. Instead of 4444 P2PKH transactions, a block can thus hold at most 2432 P2PKC transactions. However, this number comfortably exceeds the current average of 1900 transactions per block for Bitcoin Core and is thus viable.

Properties of Hardened Addresses. Hardened addresses are implemented by replacing the P2PKH pattern with P2PKC and P2SH with P2SC, respectively. This simple replacement does not change the spendability of any transaction output. We argue that hardened addresses maintain transparency and decentralization because of this property and due to the easy deployability of the scheme, which only requires assigning one of the currently reserved opcodes of `SCRIPT` the functionality of `OP_COMMIT` and adjusting the accepted output scripts. Furthermore, hardened addresses achieve a good accuracy by definition: On the one hand, they do not interfere with payments of honest users. On the other hand, hardened addresses force content inserters to resort to brute-forcing addresses as described in Section 4.1.1.1 instead of easily manipulating them. However, replacing manipulable on-chain addresses with hardened addresses increases the transaction sizes. Even though a block with

a size of 1 MB can still hold more transactions than the average Bitcoin Core block contains, precisely the per-block capacity was contentiously debated in the past [Bit15c, Mor17]. Hence, the proposed strategy is readily acceptable on a technical level, but its increased transaction sizes pose a challenge for its actual deployment.

4.2.6 Summary and Future Work

We presented three different strategies Bitcoin-like systems can adopt quickly with only little changes (**Q2**, cf. Section 1.2) to prevent unwanted content from being stored immutably on their blockchains (**Q1**). These strategies explore the available design space for such prevention strategies and complement each other: First, actively scanning for content-holding transactions and filtering them out is an intuitive strategy, but the anticipated frequent and mandatory updates to maintain up-to-date filtering rules (**P3**, cf. Section 1.1.2) raises concerns about this strategy’s effective deployability and capability to react quickly to new insertion methods (**P1**). Second, demanding mandatory minimum fees that penalize issuing large transactions promise to disincentivize the insertion of unwanted content without creating any penalty for the currently predominant forms of legitimate payment transactions. Finally, hardening on-chain addresses against manipulation via self-verifying cryptographic commitments constitutes a simple adaption of Bitcoin’s standard transaction patterns that reduces content insertion capabilities to the theoretical minimum at the price of faster growing blockchain sizes (**P2**).

However, as a result of our exploratory approach, we presented these prevention strategies mostly on a conceptual level. While we deliberately explored prevention strategies that can be deployed to retrofit existing Bitcoin-like systems with minimal technical changes, future work should investigate remaining acceptability challenges. For example, these challenges engulf defining acceptable penalties for mandatory minimum fees or trading off a blockchain’s growth rate and its improved resilience against content insertion via hardened addresses. Moreover, the detection rate of the proposed and future prevention strategies should be further evaluated.

Another important aspect of the moderation implied by prevention strategies is the handling of external services. On the one hand, prevention strategies must continue to ensure that benign applications remain unaffected by the changes they propose. On the other hand, future work should continue to explore the potential of external (and decentralized) services to further aid the prevention of unwanted content insertion, such as the distributed database of identifiers for objectionable content proposed by Cremona et al. [CTDR19] we discussed in Section 4.2.1.

Finally, prevention strategies can reduce the amount of unwanted blockchain content, but not prevent its insertion entirely (cf. Section 4.1.1.1). We thus argue that schemes allowing for the retrospective removal of blockchain content are inevitable, even though they necessarily challenge the blockchain’s immutability property and are therefore likely to impose larger required changes. As a result, in the remainder of this chapter, we investigate how Bitcoin-like blockchain systems can be extended to allow for retrospective removal of blockchain content with only little compromises regarding the blockchain’s characteristic properties.

4.3 Moderation of Blockchain Content

We have already established that permissionless blockchains are prone to the irrevocable insertion of objectionable content (**P1**, cf. Section 1.1.2) and that strategies to prevent content from entering the blockchain can provide a first line of defense, but not completely mitigate the problem. Blockchain designs hence have to enable the retrospective removal of such content to exercise full moderation capabilities (**Q1**, cf. Section 1.2) and thereby further lower the risk of negative consequences.

Redactable blockchains have been proposed to allow content to be modified or removed retroactively anywhere in the blockchain. Such modifications are publicly announced to all full nodes, which enables their operators to safely apply the modifications to their local blockchain copy while remaining capable of fully validating the (modified) blockchain. However, previously proposed redaction schemes have to trade off trust, flexibility, and performance. Previous schemes focusing on counteracting unwanted blockchain content either give the control over redactions to a fixed set of few (trusted) nodes [AMVA17, ASL19, AC22], establish transparency by relying on on-chain voting with large delays of, e.g., one week [DMT19, MZ19, TBM⁺21], or impose large per-redaction overheads [LXY⁺22].

In this section, we thus propose *RedactChain*, a *moderation framework* that enables the swift retrospective redaction of illicit content while ensuring transparency in the permissionless setting. RedactChain achieves this desirable combination by outsourcing redactions to multiple small redaction juries that are periodically elected at random. Juries coordinate and *mutually oversee* each other (**P3**) via a separate redaction log to, e.g., prevent a jury from stalling the removal of content. Each redaction jury can swiftly decide to redact reported transactions. However, its members have to cooperate, and they can only modify a small portion of the blockchain for a limited time. To account for cases where illicit content remained undetected for a long time, RedactChain can still use a long-term voting scheme (e.g., [DMT19]) as a fallback mechanism. In our design, we rely on chameleon hash functions (CHF) [KR00, AM05, AMVA17], which enable anybody knowing a secret trapdoor key to modify individual blocks efficiently. In contrast to other CHF-based approaches, RedactChain does not rely on a fixed CHF controlled by a fixed set of redactors. Instead, newly elected juries establish new CHF via distributed key generation (DKG) [GJKR07]. Further, we subject all redactions to rules that enable juries to redact content even from spendable transaction outputs without permitting arbitrary modifications, e.g., the rules prevent attempts to override previous payments. Finally, the redaction log provides a transparency ledger of all accepted modifications, which additionally supports the coordination between currently active redaction juries.

While the prevention strategies we presented in Section 4.2 had the goal of being easily retrofitted to established systems, with RedactChain we explore how slightly weakening this requirement (**Q2**) creates additional opportunities to counteract unwanted content (**Q1**). Our results show that RedactChain can realize swift and transparent redactions in the permissionless setting while keeping the overhead due to its transparency feasible (**P2**). We provide the source code for our prototypic implementation of RedactChain as open source under the GPL-3.0 license [MAP⁺22b].

In the following, we first discuss previous redaction schemes (Section 4.3.1) to motivate the need for swift and transparent redactions (Section 4.3.2). We then give a brief introduction to relevant cryptographic building blocks (Section 4.3.3). Afterward, we give an overview of RedactChain’s design (Section 4.3.4) before detailing its handling of reports of content-holding transactions (Section 4.3.5), the decentralized redaction process executed by a redaction jury (Section 4.3.6), and the coordination between redaction juries (Section 4.3.7). Next, we present our evaluation (Section 4.3.8) and then conclude with a short discussion and summary (Section 4.3.9).

4.3.1 Related Work

Previous approaches have acknowledged that unconditionally immutable blockchains are prone to consequences related to objectionable content, and they proposed designs for redactable blockchains. In the following, we give an overview of past designs of redactable blockchains; thereby, we complement our discussion of mitigation schemes that do not question unconditional blockchain immutability from Section 4.2.1. We identify three main categories of redactable blockchains and consider the redaction schemes to be either *trust-based*, *voting-based*, or *policy-based*.

Trust-based Redaction Schemes

The concept of redactable blockchains was introduced first in 2017 by Ateniese et al. [AMVA17] as an alternative blockchain design that allows to redact transactions retrospectively. Their design makes use of *chameleon hash functions (CHF)* [KR00, AM05] instead of traditional cryptographic hash functions to interlink the individual blocks. As we illustrate further in Section 4.3.3.1, these hash functions enable a dedicated party knowing a private key associated with the CHF to efficiently compute collisions, which, in turn, enables them to replace blocks at any point in the blockchain. Hence, this party effectively becomes a blockchain *moderator* or *redactor*. In this setting, the system’s CHF is fixed during the blockchain’s inception, i.e., the private key does not change over the course of its lifetime.

While the moderator can react quickly to remove content, this line of redactable blockchains has the downside that the moderator is inherently trusted to conduct redactions faithfully. Namely, designating a single blockchain moderator is in conflict with the decentralized approach of permissionless blockchains such as Bitcoin. Previous works either applied threshold cryptography to decentralize redactions [AMVA17, ASL19] or improved the transparency of redactions [AC22].

As we discuss further in Section 4.3.3.2, threshold cryptography enables executing protocols involving cryptographic building blocks in a decentralized manner such that credentials can be split across multiple independent parties who then need to cooperate to use these credentials. In the context of redactable blockchains, the private key required to conduct redactions can be split across multiple parties instead of a single moderator [AMVA17, ASL19]. However, previous approaches struggle to identify a viable moderator set in the permissionless setting [AMVA17, ASL19].

Even when decentralizing redactions by defining a moderator set, these changes are not accountable. The moderators can alter blocks arbitrarily, which means that they also become responsible for manually keeping the transaction graph intact, i.e., they may never redact already spent transaction outputs even though they are given the technical means to do so. Finally, previous CHF-based redactable blockchains do not allow for joining nodes to learn whether a block has been redacted in the past, since CHFs are specifically designed to keep the hash value intact despite any (authorized) redaction. Recently, Astrizi and Custódio [AC22] presented a design for a redactable blockchain that enables blockchain observers to identify whether a given block was redacted in the past, albeit in a centralized setting.

Takeaway. Trust-based redaction schemes promise a near-instantaneous removal of unwanted content, but their current instantiations raise questions regarding their decentralization and public verifiability.

Voting-based Redaction Schemes

To mitigate strong trust assumptions regarding the moderation of blockchain content, other designs for redactable blockchains rely on *public voting* instead of fixed redactors. An initial voting-based redaction scheme for the permissionless setting was proposed by Deuber et al. [DMT19]. In this scheme, any user can report a block for containing content that should be redacted and propose an updated version for that block. Following this request, the miners indicate in their blocks whether they are in favor of the proposed redaction or whether they oppose it. After the voting period concludes, all miners and full nodes can evaluate whether a tunable majority is in favor of accepting the proposed redaction and update their local blockchain copy accordingly. In the blockchain proposed by Deuber et al., the validity of the interlinking of blocks can be asserted in either of two ways, depending on whether a given block was edited or not. Unedited blocks are validated in the same way as for traditional blockchains, i.e., a block’s hash value must correspond to the back link stored in the block’s successor. However, any block modification invalidates this back link; for handling this, the block’s original hash value is stored alongside the block header. If the traditional check fails, a full node can still check whether (a) the back link would be valid for the block’s original state, (b) the new block is valid, and (c) the new block has been accepted as a result of the subsequent voting period. In this case, the block is a valid modified version of an originally accepted block.

When applied to Bitcoin, the authors propose to accept a redaction using a simple-majority policy and a voting period of 1024 blocks [DMT19]; this voting period corresponds to roughly one week of wall-clock time given Bitcoin’s expected inter-block interval of ten minutes (cf. Section 2.4.1). Furthermore, the approach defines specific policies for valid redaction operations to mitigate the harmful redactions allowed in the context of trust-based redaction schemes. Most notably, the scheme only allows removing provably unspendable transaction outputs [DMT19]. This restriction essentially boils down to redacting only `OP_RETURN` transaction outputs, which covers only one of multiple available content insertion methods, as we analyzed in Section 3.3.

Subsequent voting-based redaction schemes improved different aspects of this initial design. Reparo [TBM⁺21] is a closely related approach that focuses on retrofittable compatibility with already established blockchain systems, i.e., also older content should become redactable after the fact. The scheme’s proposed Bitcoin integration ultimately relies on the voting scheme of Deuber et al. [DMT19], which results in likewise large delays and limited redactability. Marsalek and Zefferer [MZ19] further keep track of redaction metadata for increased verifiability and ensure that standard transaction outputs, such as P2PKH, are redactable. Li et al. [LXY⁺22] presented an approach where redactions are performed by smaller committees, who only have temporal redaction capabilities; however, as of now, this delegation of control comes with considerable per-redaction overheads of roughly 110 kB [LXY⁺22]. Finally, and concurrently to our work, Meng et al. [MNLX22] proposed a round-based redaction scheme where committees are elected ad-hoc using Algorand’s algorithm [CM19]; even though this approach promises fast decisions on pending redactions, it suffers from the same policy limitations as other previous works.

Takeaway. Voting-based redaction schemes constitute a suitable adaption for content moderation in the permissionless setting, but they achieve their decentralization and public verifiability at the cost of slower responses, limited redaction capabilities, or larger per-redaction overheads.

Policy-based Redaction Schemes

The final flavor of redaction schemes are *policy-based*. These redaction schemes primarily center around GDPR-related risks and focus on the self-sovereignty of the transaction creator, i.e., the transaction creator should be in control of redactions of their own transactions. To this end, Derler et al. [DSSS19] proposed a redaction scheme where the transaction creator can express who is allowed to alter a transaction using a combination of *ciphertext-policy attribute-based encryption (CP-ABE)* [BSW07] and *chameleon-hashes with ephemeral trapdoors (CHETs)* [CDK⁺17]. Essentially, this scheme enables transaction creators to optionally specify a policy over attributes assigned to other nodes of the blockchain network to define who is capable of modifying blockchain content at the granularity of individual transactions. This and related approaches [PDC17, DKJ19, LYOK19, DSSS19, TLL⁺20] have the main goal to empower the transaction creator to exercise fine-grained control over their transactions. However, these approaches cannot protect against users who insert illicit content, as the transaction owner has to cooperate to make their content redactable. Recently, this concept has been further evolved by making redactions on the transaction level accountable [Luo22, XHY⁺23] and by reinstalling a centralized moderator who can oversee all redaction activities [JSZ⁺21].

Takeaway. Considering the threat of objectionable blockchain content insertion, policy-based redaction schemes that let the transaction creator decide whether and how their transactions may be altered are not well-suited to fulfill this task. Even though we notice a first approach that combines user-centered modifications and moderator-based redactions [JSZ⁺21], this approach currently reintroduces concerns regarding the moderator’s trustworthiness in the permissionless setting.

Summary. Various designs for redactable blockchains have been proposed in the past. However, each approach has distinct shortcomings when it comes to retrospectively removing illegal blockchain content: While purely policy-based schemes are not applicable in this scenario, trust-based schemes involve an undesirable centralization of moderation capabilities. Voting-based schemes are designed to maintain decentralization, but they require lengthy voting periods. Permissionless blockchains would, thus, benefit from a decentralized redaction scheme that allows for a fast reaction without compromising on transparency, as we detail in the next section.

4.3.2 Missing Swift and Transparent Redactions

After elaborating the risks of unwanted content insertion in permissionless blockchains and showing that content is actively being inserted in Chapter 3, we have argued in Section 4.1.1.1 that the practice cannot be prevented reliably. As a result, the prevention strategies we presented in Section 4.2 provide a first line of defense against uncontrolled content insertion, but blockchain systems must be designed with redaction capabilities in mind to be able to cope with the remaining risks.

Since any unwanted blockchain content is distributed to all full nodes in the permissionless setting, respective designs have to emphasize a swift response to identified content to limit the potential damage as early as possible. At the same time, required redactions must be transparent to all established and future full nodes, such that redactions do not undermine the public verifiability that is desirable for permissionless blockchains. Ultimately, the goal must be to enable swift redactions without reducing trust in the network’s commitment to uphold the blockchain’s overall immutability during normal operation, i.e., when no redactions are currently required.

Unfortunately, our analysis of previously proposed redaction schemes shows that these approaches do not provide reaction capabilities that are swift *and* transparent at the same time. Furthermore, previous approaches cannot appropriately cover content added via unintended insertion methods, such as address manipulation. As we concluded in Section 4.3.1, all current variants of redaction schemes have shortcomings regarding the scenario of unwanted content insertion: Trust-based redaction schemes reintroduce a point of centralization and do not provide transparency because either a single redactor is empowered to alter any part of the blockchain or the approaches are not explicitly forcing the (potentially distributed) redactor to be transparent about their reason for executing a particular redaction. Contrarily, voting-based redaction schemes either react only slowly or create considerable per-redaction overheads. Finally, policy-based redaction schemes require the transaction creator to cooperate, which renders them inapplicable in our scenario.

These previous approaches further focused on the technical enablers for redactable blockchains. Other crucial aspects tend to remain unaddressed: Potentially spendable transaction outputs can hold content (e.g., via address manipulation), but they are either explicitly declared non-redactable [DMT19, TBM⁺21] or not considered in prior designs. However, we have shown in Section 3.5 that address manipulation has been used in the past to insert also questionable content. While we

noticed a shift toward using the large `OP_RETURN` fields offered by Bitcoin SV, this insertion method prevails and must be accounted for by redaction schemes. However, modifying transaction outputs may affect the validation of future transactions attempting to spend outputs of affected transactions; hence, transaction outputs require special treatment when making them redactable. Another neglected side effect of altering transactions to remove unwanted content is changing transaction identifiers. As blockchain systems identify transactions based on their hash value (cf. Section 2.3.1), these identifiers are prone to change with every modification of the transaction. Hence, redaction schemes must consider situations where a transaction with spendable outputs is referenced after it has been modified, i.e., after the transaction identifier changed.

Hence, the goal of this contribution is to develop a redaction scheme allowing to moderate blockchain content in the permissionless setting that satisfies the following goals, which we derive from both our general requirements for mitigating blockchain content (cf. Section 4.1.1.2) and our discussion of related work:

(G1) Swift Reaction. A moderation scheme must enable the *swift redaction* of identified illicit blockchain content, i.e., honest nodes must be able to promptly delete such content without further implications for the network’s operability.

(G2) Transparent Decisions. Simultaneously, the permissionless setting requires that moderation schemes are *transparent* regarding *which* actions have been taken and *why*. Established as well as joining nodes must be able to easily validate the full blockchain history and the full transaction graph regardless of past redactions while being able to assess the reasoning for redactions and potential disputes. For instance, the most current version of each block must be easily identifiable.

(G3) Decentralization. Redaction schemes must account for the *decentralized* nature of permissionless blockchains, even when ensuring swift and transparent redactions. Consequently, redaction powers must not be held by a fixed node or one group of nodes exclusively.

(G4) Disputes and Finality. Whether or not content is unwanted or objectionable is oftentimes debatable. Proper moderation schemes must *facilitate dispute mediation* but *ultimately finalize decisions* to ensure that consensus about each block’s state is reached eventually, despite potential redactions.

(G5) Well-Defined Rule Set. The scheme must provide an explicit rule set for *how* to alter any given transaction. Namely, the defined rule set must go beyond only redacting provably unspendable transaction outputs and also consider, for instance, address manipulation. In doing so, the redaction scheme must prevent undesirable side effects, such as redacting already spent transaction outputs.

(G6) Scalability. Our scheme must ensure swift redactions and acceptable overheads for achieving transparency and coordination also for large network sizes and large numbers of content reports.

In the following, we first give additional technical background on suitable building blocks for closing this gap and achieving these goals before detailing the design of our proposed redactable blockchain system.

4.3.3 Cryptographic Building Blocks

After having outlined the need for enabling swift and transparent redactions for permissionless blockchains, we now give an overview of the cryptographic building blocks our proposed solution relies on. First, we give an introduction to chameleon hash functions (Section 4.3.3.1), which were already used in previous work to enable a dedicated redactor to efficiently replace blocks in the blockchain based on the knowledge of a private key [AMVA17]. Second, we present building blocks from the domain of threshold cryptography (Section 4.3.3.2) that are suitable to distribute control in cryptographic protocols among multiple, independent parties.

4.3.3.1 Chameleon Hash Functions

Blockchain systems make heavy use of cryptographic hash functions, most notably to cryptographically interlink the individual blocks that make up the blockchain (cf. Section 2.2): Due to the second pre-image resistance of cryptographic hash functions, an attacker is highly unlikely to be able to modify a block somewhere in the middle of the blockchain without changing the block’s hash value and, thereby, invalidate the blockchain’s integrity. *Chameleon hash functions (CHF)* [KR00] are a special class of hash functions that behave just like cryptographic hash functions on the surface. However, a CHF is parametrized with a secret *trapdoor key* that is only known to predetermined authorized entities. Any entity knowing the secret trapdoor key can now, in contrast to traditional cryptographic hash functions, efficiently compute collisions for any given (chameleon) hash value. Consequently, CHFs have been used to realize redactable-blockchain designs in the past [AMVA17, ASL19, HZM⁺19, HZM⁺20].

In the following, we give an overview of a class of CHFs proposed by Ateniese and de Medeiros [AM05], which was subsequently used by Ateniese et al. [AMVA17] to implement one of the earliest redactable blockchains. We first give an overview of how CHFs facilitate the redaction of blockchain information before presenting the construction of the CHFs proposed by Ateniese and de Medeiros [AM05]; finally, we outline how these CHFs enable the efficient computation of hash collisions.

Overview. From now on, we focus on the CHF’s application to redactable blockchains and loosely base our presentation on the notation used by Ateniese et al. [AMVA17] to describe their redactable blockchain. In this setting, the consensus rules fix a CHF at the time of the blockchain’s inception, and a dedicated *redactor* knows the CHF’s corresponding trapdoor key. During normal operation, the miners attempt to solve the PoW puzzle using the CHF instead of a traditional cryptographic hash function. Once the redactor is made aware of information to be removed from a block already included in the blockchain, they update the affected block accordingly and then use the trapdoor key to replace the block with its updated version. Hence, we also refer to the secret trapdoor key as the *redaction key* due to its crucial role in enabling redactions.

The CHF proposed by Ateniese and de Medeiros [AM05] bases on Nyberg-Rueppel signatures [NR95] and provides the following functionality for realizing redactable

blockchains: Given a group \mathbb{G} for which the discrete logarithm problem is hard and a generator g , the redactor initializes the mining process for a new blockchain by creating a key pair (k, g^k) , which consists of the *redaction key* k and the *mining key* g^k , and sharing g^k with the miners. The miners then use the mining key to solve the PoW puzzle as for traditional blockchain systems. For this purpose, they use a *hashing algorithm* $\mathcal{H}(\cdot)$ to compute the redactable hash value $\mathcal{H}(B, g^k) = (h, \xi)$ for a given block B . Here, h is the hash value of the block and ξ is a random *check value* stored alongside the block. The check value enables the redactor to subsequently update the block without altering its hash value h . To this end, the redactor uses a *collision-finding algorithm* $\mathcal{C}(B', (B, h, \xi), k) = \xi'$ to efficiently compute a new check value ξ' that constructs a collision between the original block B and an updated block B' . Anybody can use a *verification algorithm* $\mathcal{V}(B, (h, \xi), g^k)$ to ascertain the correctness of h for the initial as well as any updated version of a block. In case of a redaction, the original block B and check value ξ have been replaced with their updated versions B' and ξ' , respectively, such that the verification algorithm $\mathcal{V}(B', (h, \xi'), g^k)$ still confirms that h is a valid hash value also for (B', ξ') .

Setup. The CHF scheme of Ateniese and de Medeiros works as follows [AM05]. System parameters of the scheme are primes p and q , where $p = 2q + 1$, and a generator g of the subgroup of quadratic residues of \mathbb{Z}_p^* , which has an order of q . Furthermore, the CHF utilizes a collision-resistant hash function for its construction; in the context of this work, we use SHA256 for this purpose. To set up a CHF, the redactor chooses a random and secret redaction key $k \in [1, q - 1]$ and shares the corresponding public mining key g^k with the miners.

Hashing Algorithm. After the setup phase, the miners use the following hashing algorithm $\mathcal{H}(\cdot)$, which involves the public mining key g^k , when attempting to solve the PoW puzzle to find a new block:

$$\mathcal{H}(B, g^k) = r - (g^{\text{SHA256}(B\|r) \cdot k} \cdot g^s \bmod p) \bmod q$$

During this process, the miner chooses the two values $r, s \in \mathbb{Z}_q$ at random. These values constitute the check value $\xi = (r, s)$, which enables the redactor to efficiently redact information using the collision-finding algorithm described below.

Collision-Finding Algorithm. The redactor can efficiently update a block B with an updated version B' by computing a collision (B', ξ') for the initial block and check value (B, ξ) . To do this, the redactor chooses a random *collision value* $a \in [1, q - 1]$ and computes $\xi' = (r', s')$ based on the block's current hash value $\mathcal{H}_{r,s}(B, g^k)$:

$$\begin{aligned} r' &= \mathcal{H}_{r,s}(B, g^k) + (g^a \bmod p) \bmod q \\ s' &= a - \text{SHA256}(B'\|r') \cdot k \bmod q \end{aligned}$$

Afterward, the redactor disseminates B' and ξ' to all nodes in the network. The redactor must be careful to keep a secret because the block's original hash value $\mathcal{H}_{r,s}(B, g^k)$, the new block B' , and the new check value $\xi' = (r', s')$ will be publicly known. Anyone knowing a chosen collision value a could reconstruct the secret redaction key k using the equations above. Similarly, reusing a for multiple redactions would leak k as well, as knowing that two check values $\xi' = (r', s')$ and $\xi'' = (r'', s'')$ were obtained via the same collision value a yields that $a = \text{SHA256}(B'\|r') \cdot k - s' = \text{SHA256}(B''\|r'') \cdot k - s''$, i.e., $k = (s' - s'') \cdot (\text{SHA256}(B'\|r') - \text{SHA256}(B''\|r''))^{-1}$.

Verification Algorithm. When receiving a block B , all nodes can verify the block's hash value h under the CHF using the following verification algorithm $\mathcal{V}(\cdot)$: Each node first reads $\xi = (r, s)$ from the block's header and computes $\mathcal{H}_{r,s}(B, g^k)$, i.e., each node reruns $\mathcal{H}(\cdot)$ for a now-fixed check value $\xi = (r, s)$. The result of the verification algorithm $\mathcal{V}(B, (h, (r, s)), g^k)$ then yields **true** if and only if $\mathcal{H}_{r,s}(B, g^k) = h$ holds.

Validity of Collisions. After having presented the details of the CHFs due to Ateiese and de Medeiros [AM05], we briefly recapitulate why their construction yields valid hash collisions [AM05]: After using the collision-finding algorithm on the original block and check value (B, ξ) and the updated block B' , the redactor obtains a new updated check value ξ' . As the verification algorithm involves recomputing the hash value after fixing (r, s) , a collision is valid if $\mathcal{H}_{r',s'}(B', g^k) = \mathcal{H}_{r,s}(B, g^k)$ holds. When considering how h and (r', s') are computed by the hashing and collision-finding algorithms, and by substituting from those equations accordingly, as well as substituting $e = \text{SHA256}(B' || r')$ for readability, we observe that this equality indeed holds as reported by the authors [AM05]:

$$\begin{aligned}
 h &= \mathcal{H}_{r',s'}(B', g^k) = r' - (g^{ek} \cdot g^{s'}) \bmod p \bmod q \\
 &= r' - (g^{ek} \cdot g^{a-ek \bmod q}) \bmod p \bmod q \\
 &= r' - \left(\frac{g^{ek}}{g^{ek}} \cdot g^a \bmod p \right) \bmod q \\
 &= r' - (g^a \bmod p) \bmod q \\
 &= \mathcal{H}_{r,s}(B, g^k) + (g^a \bmod p) - (g^a \bmod p) \bmod q \\
 &= \mathcal{H}_{r,s}(B, g^k)
 \end{aligned}$$

Hence, the construction of $\xi' = (r', s')$ for obtaining B' using the random collision value a updates the block without altering its hash value.

Applicability to Redactable Blockchains. This class of CHFs proves beneficial for the envisioned application of realizing redactable blockchains for two reasons. First, these CHFs prevent key exposure [AM05], i.e., third parties cannot derive the secret redaction key from observing a collision $(B', (h, \xi'))$ with respect to previously recorded information $(B, (h, \xi))$. This property is especially important due to the desired transparency of the underlying blockchain system. We have to assume that some nodes observe the initial version of a block and then receive an update in case of a necessary redaction. In this case, the nodes must not be able to learn about the secret redaction key, as that knowledge would empower them to redact previously recorded information anywhere in the blockchain themselves. Second, the collision-finding algorithm can easily be transformed into a distributed variant relying on threshold cryptography, which we give an overview of in the next section. As a result, we can distribute the control over redactions to a set of independent redactors instead of relying on a single redactor all nodes would have to trust; we detail our construction for the distributed CHF used in our proposed redactable blockchain as part of its design in Section 4.3.6.1.

4.3.3.2 Threshold Cryptography

Threshold cryptography comprises the application of principles known from secure multiparty computation (SMC) to create decentralized cryptographic protocols, where the control over credentials is distributed among independent entities who have to cooperate as a result. In this section, we provide a brief overview of the building blocks that we use in our design to distribute the control over redactions across multiple independent parties.

Secure Multiparty Computation and Secret Sharing. A secret-sharing scheme allows a *dealer* to *share* a secret value s among n players such that any $t + 1$ players can jointly *recombine* s , where t is a configurable threshold, and any set of at most t players learns nothing about s . Shamir proposed one of the most widely applied secret-sharing schemes to accomplish this behavior [Sha79]: Using Shamir’s scheme, the secret value is an integer $s \in \mathbb{Z}_p$, where p is a large prime number, and the dealer picks a random polynomial $f(x) = \sum_{i=0}^t a_i x^i$ of degree t with $s = a_0 = f(0)$ being the secret value and sends each player p_i its secret share $[s]_i = f(i)$ ($i = 1, \dots, n$) in private. During the recombination step, every player broadcasts their share to every other player. Any player knowing at least $t + 1$ shares can then successfully reconstruct $f(x)$ using Lagrange interpolation and subsequently compute $s = f(0)$ to obtain the secret value.

Shamir’s secret-sharing scheme especially facilitates secure computations among multiple parties because its secret shares are *additively homomorphic*. This means that players can locally add their shares of different secrets to gain a share of the sum of both secret values, i.e., $[s_1]_i + [s_2]_i = [s_1 + s_2]_i$, and they can multiply shares by a constant, i.e., obtain $c \cdot [s]_i = [c \cdot s]_i$ [BOGW88]. Similarly, players can interactively multiply shares at the cost of additional communication between all players, which renders share multiplication costly compared to the local share addition [BOGW88]. The main problem of share multiplication is that simply multiplying a player’s shares $[s_1]_i$ and $[s_2]_i$, which are derived using polynomials $f_1(x)$ and $f_2(x)$, respectively, would yield a secret share $[s_1 \cdot s_2]_i$ for a combined polynomial $f_1(x) \cdot f_2(x)$, but that polynomial has degree $2 \cdot t$ instead of t [BOGW88]. Before continuing with further computations, the players thus engage in an interactive protocol to jointly derive a new polynomial with degree t and the same secret value as $f_1(x) \cdot f_2(x)$, and they ensure that the players derive valid secret shares according to that polynomial [BOGW88]. Theoretically, any functionality can be computed using this framework as long as at most $t < n/3$ players are controlled by a malicious adversary [BOGW88] and no invalid shares are used to recombine a secret value. A player who received more than $2 \cdot t$ shares can use the Welch-Berlekamp algorithm [WB86] to locally identify invalid shares and thus exclude them from further computations.

Reliable Broadcast Channels. Reliable broadcasts enable players in a distributed setting to detect equivocation, e.g., when players broadcast their shares to recombine a secret. Bracha [Bra84] presented a simple protocol to implement a reliable broadcast that tolerates up to t out of n players to be corrupted by a malicious adversary as long as $t < n/3$. The players reach consensus on whether to accept or reject a message M sent by an initiator by broadcasting M to all other players alongside

their local state, which is either `echo` or `ready`. The initiator first sends M to all players, who then enter the `echo` state, i.e., each player p_i broadcasts (M, echo_i) to all other players. More precisely, a player enters the `echo` state by echoing the first valid message it receives to be able to compensate for an untrustworthy initiator who sends M only to some but not all players or who equivocates, i.e., sends M to some players and a different message M' to others. When a player is subsequently convinced that the honest players will reach consensus to accept M , it enters the `ready` state and broadcasts (M, ready_i) to all other players. The player is convinced of M being broadcast correctly if it either receives at least $(n + t)/2$ `echo` messages or at least $t + 1$ `ready` messages from other players. The player then accepts M if it receives at least $2t + 1$ `ready` messages. Ultimately, all honest players will take the same decision to either accept M or discard it at some point, thereby implementing the desired broadcast functionality in a distributed setting.

Distributed Key Generation. Threshold cryptography heavily relies on cryptographic keys being distributed across the players such that no single player can abuse, for instance, a shared private key. *Distributed key generation (DKG)* enables the players to securely generate a public-private key pair (k, g^k) in a distributed manner, i.e., each player holds a share $[k]$ of a distributed private key k and all players obtain the same corresponding public key g^k . Gennaro et al. [GJKR07] presented a DKG protocol for groups where the discrete logarithm problem is hard that is secure as long as at most $t < n/2$ players are corrupted by a malicious adversary. This protocol operates in two phases: First, the players generate their shares $[k]$ of the private key in a distributed manner. The protocol ensures that k is drawn uniformly at random [GJKR07]. Second, the players jointly recombine the corresponding public key g^k . Gennaro et al. utilize different *verifiable* secret sharing schemes [Fel87, Ped91] to identify and exclude equivocating players. This way, the honest players ensure that they compute their private values only based on data that has been shared consistently among the players. Hence, DKG constitutes a valuable building block for distributed protocols that rely on cryptographic operations.

Based on these building blocks, we can now present the design of our proposed moderation framework involving a redactable permissionless blockchain, RedactChain.

4.3.4 RedactChain Overview

We start our presentation of RedactChain by first giving an overview of its components, moderation process, and block structure before presenting the individual redaction steps in detail in the following sections.

Moderation Process

Figure 4.6 shows how RedactChain’s components interact to moderate and execute redactions in a swift and transparent manner in four steps (Steps ①–④):

Any user can ① report a transaction for illicit content, including a claim arguing for the redaction, to one of m simultaneously active *juries* J_i with n members each.

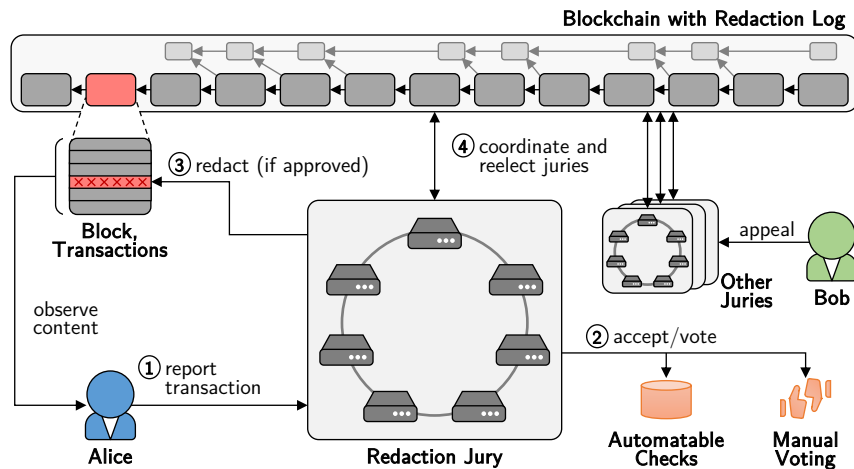


Figure 4.6 RedactChain operates in four steps. Alice ① reports transactions containing illicit content to a jury, which ② validates the claim and ③ redacts it if warranted. Juries ④ coordinate so that Bob can appeal the decision. RedactChain replaces juries periodically.

The jury then ② votes off-chain whether or not to redact the reported transaction and ③ executes the redaction depending on the outcome of the voting process. Similarly to previous approaches [AMVA17, ASL19, AC22], RedactChain uses chameleon hash functions (CHF) (cf. Section 4.3.3.1) to grant juries the ability to redact blocks. However, RedactChain limits the juries’ influence in multiple ways to prevent a misuse of power. First, we rely on distributed key generation (DKG) (cf. Section 4.3.3.2) to distribute the control over the secret redaction key across the jury members. RedactChain further defines strict rules for valid modifications. For instance, each jury may modify any transaction at most once, and other nodes will reject further modifications once the transaction has not been modified for Δ_R blocks to ensure that disputes are ultimately settled. New juries are elected every Δ_D blocks from the recently successful miners, i.e., replaced juries can no longer modify new blocks. To prevent older content from becoming non-redactable, RedactChain can fall back onto slower public voting processes (e.g., [DMT19]). Furthermore, in contrast to previous solutions, RedactChain also supports the global redaction of theoretically spendable outputs (e.g., manipulated P2PKH transactions) by *obfuscating* instead of deleting those outputs. Since the spendability of such transactions is not affected by the obfuscation step, juries can execute such redactions without having to inspect content inserted via unintended means manually.

Juries ④ coordinate via a separate *redaction log*. The redaction log has three functions. First, juries record *update entries* on the redaction log to document their modifications, i.e., redactions or restorations. Second, the redaction log is coupled to the main blockchain to establish consensus about an approximate timing of all events. Nodes maintain per-transaction *redaction timers* $\Delta_{R,t}$ based on this approximate timing. The redaction timer $\Delta_{R,t}$ of transaction t gets reset whenever a valid modification of t is recorded on the redaction log, and it expires when it was not reset for Δ_R blocks. Nodes reject any modification proposed after $\Delta_{R,t}$ expires, i.e., when a superseded jury attempts to redact finalized blocks. This way, RedactChain prevents juries from overstepping their competencies and establishes transparency.

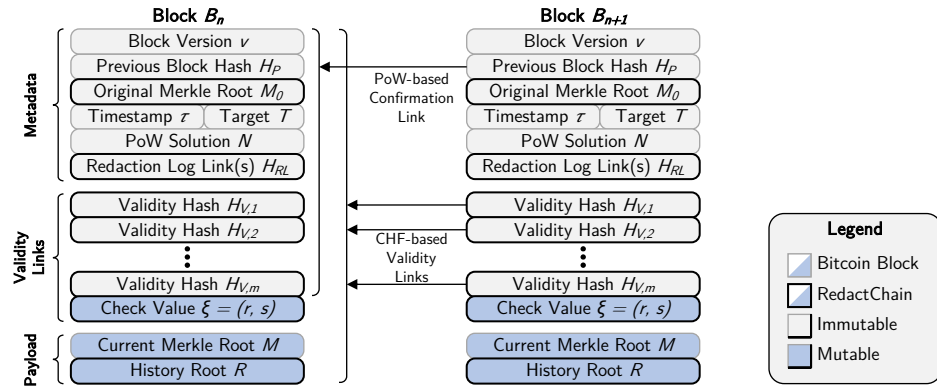


Figure 4.7 To enable modifications, a RedactChain block enhances the traditional Bitcoin block header and keeps track of the original Merkle root (M_0), i.e., the block’s state prior to any modifications, one CHF-based validity link per redaction jury ($H_{V,i}$), and the check value ($\xi = (r, s)$) as used by the CHFs. Additionally, the block header includes references to the block’s redaction history (R) and one or more links confirming entries of the redaction log (H_{RL}) to ensure transparency and enable coordination among the redaction juries.

Further, users can now appeal to unwarranted redactions by showing the original transaction to another jury. Finally, new juries announce their new CHF’s public values on the redaction log so that miners can start using those CHFs.

Blockchain Structure

Figure 4.7 shows the structure of RedactChain blocks compared to those of Bitcoin. RedactChain uses one *validity link* $H_{V,i}$ per jury to assert the previous (potentially redacted) block’s current state, in addition to the common PoW-based *confirmation link* H_P . The confirmation link only covers the immutable fields of a block header. Each block holds an immutable copy of its *initial* Merkle tree root M_0 to be covered by the next block’s confirmation link. Additionally, each block has m validity links corresponding to the m active juries’ CHFs. By checking the validity links, any node can assert that the block is in its initial state or that a responsible jury has redacted it. Namely, jury J_i redacts a block by updating its check value $\xi = (r, s)$ via its CHF’s trapdoor key in a decentralized manner (cf. Section 4.3.6.1) to keep $H_{V,i}$ of the next block intact. Finally, the history root R accumulates all update entries corresponding to modifications of transactions within the current block (cf. Section 4.3.6.3).

4.3.5 Detecting Unwanted Blockchain Content

In this section, we present how users report transactions for holding unwanted content to a redaction jury (Step ①, Section 4.3.5.1) and how juries handle these reports (Step ②, Section 4.3.5.2), i.e., we present their decision-making process leading to the result of either redacting the transaction or discarding the report.

4.3.5.1 Filing Reports

Similarly to previous solutions [DMT19, TBM⁺21, LXY⁺22], RedactChain enables users to report transactions containing illicit content. Besides normal users, established organizations fighting illicit content, such as the Internet Watch Foundation (IWF) [Int15a], can monitor the blockchain and report unwanted and objectionable content in a timely manner. A user files their report by contacting a randomly selected jury based on the members' host names, ports, and public keys published as part of the jury election process (cf. Section 4.3.7.1). Each report consists of a transaction's identifier and a succinct claim of why the content should be redacted. RedactChain relies on the reliable-broadcast primitive due to Bracha [Bra84] (cf. Section 4.3.3.2) for filing reports to ensure that all jury members receive the same report.

4.3.5.2 Handling Reports

After receiving a report, the jury engages in an off-chain decision-making process to determine whether to redact the reported transaction. Here, we distinguish content inserted via unintended and intended methods, as they have distinctly different characteristics (cf. Section 3.3.1). Namely, intended methods typically only allow for storing small data chunks and unintended methods can circumvent these restrictions by interfering with the validation of financial transactions.

We first consider handling unintended insertion methods, where address manipulation is the predominant method for unwanted content insertion. Since falsely removing spendable transaction outputs at the suspicion of address manipulation could alter the transaction graph, RedactChain is careful to not remove affected outputs during the redaction but instead only obfuscates them (cf. Section 4.3.6.2). Thus, juries can automate the decision-making process and always redact transactions reported for embedding content via unintended means because this form of redaction does not affect the outputs' spendability. However, the obfuscation inflicts overheads. Combining this automated obfuscation with further heuristics from our discussions of prevention strategies (cf. Section 4.2), e.g., requiring a minimum number of transaction outputs or explicitly checking for manipulation using content detectors, can help identify false reports and reduce these overheads.

Contrary to the example of address manipulation, intended insertion methods, by definition, are not abusing blockchain features. Hence, the jury members have to manually vote for or against redacting corresponding content. We expect only a few instances of such manual voting, since encoding objectionable content is harder due to these methods' limited capacity. In scenarios where the insertion of larger data chunks is desired (e.g., via the Bitcoin Data Protocol, cf. Section 3.3.2), jury members need more support. For example, public access to fingerprint databases of known illicit material could vastly simplify the safe and automated detection of such material [CTDR19]. Alternatively, stakeholders such as the IWF, which already works with such databases to fight child abuse on the Internet [Int15a], could monitor the blockchain for such content.

4.3.6 Decentralized Redaction Process

After approving a report, a redaction jury needs to implement the redaction on the blockchain. We now detail this redaction process (Step ③) and, particularly, how juries apply decentralized CHF's during the redaction (Section 4.3.6.1), how they alter transactions (Section 4.3.6.2), and how they ensure that redactions are transparent to all nodes (Section 4.3.6.3).

4.3.6.1 Threshold CHF's for Trusted Redactions

RedactChain relies on distributed CHF's during its redaction process. This way, redaction juries can modify the blockchain efficiently without increasing the complexity for the other nodes when they validate the integrity of the updated blockchain. Whenever a new jury is elected (cf. Section 4.3.7.1), its members first establish and announce a new CHF before they can jointly apply that CHF to execute redactions. In the following, we present the decentralized CHF establishment and its application involving the decentralized computation of collisions in more detail.

CHF Establishment. RedactChain makes use of the class of CHF's we presented in Section 4.3.3.1 that was also used in the initial trust-based redaction scheme by Ateniese et al. [AMVA17]. To recap, these CHF's use a *check value* $\xi = (r, s)$ to enable efficient collisions. This check value is chosen at random during the initial hashing process but enables the redactor to modify the block using a random *collision value* a : This collision value can be used together with the secret redaction key to compute a new check value such that the block's hash value does not change despite the redaction. The design by Ateniese et al. [AMVA17] provides fast redactability, but it relies on fixing one CHF at the inception of the blockchain system, which implicitly fixes the role of the redactor. To eliminate the dependency on a fixed redactor, RedactChain dynamically exchanges the currently active CHF's over time: During the bootstrapping of a new redaction jury (cf. Section 4.3.7.1), its members first establish connections to each other and jointly create a new CHF. The jury uses DKG (cf. Section 4.3.3.2) to generate a new distributed redaction key, i.e., each member only holds a share of this secret key, but all members obtain the corresponding public mining key required to compute hash values. The jury then publishes a *jury assembly block* containing the mining key on the redaction log (cf. Section 4.3.7.1) so that miners can start mining blocks using the new CHF.

Secure Collision Computation. The jury can now redact transactions from blocks mined during their duty period by jointly computing a collision for their CHF. The initial design by Ateniese et al. proposed to decentralize the redaction process by distributing shares of the redaction key among the redactors using DKG and then recombining the secret redaction key before computing the collision [AMVA17]. However, no jury member should learn the redaction key k in the clear, as they would then be able to issue arbitrary redactions from then on. RedactChain instead makes full use of threshold cryptography (cf. Section 4.3.3.2) to decentralize the collision computation among the jury members. Specifically, individual jury members neither learn the redaction key k nor the collision value a , which could be used to reconstruct

k (cf. Section 4.3.3.1), in the clear. Our distributed variant of the collision-finding algorithm achieves this by having the jury members engage in another round of DKG to randomly draw the collision value in a decentralized manner; as a result, each jury member holds a share $[a]$ of the collision value and knows g^a . With this information, each jury member can locally compute r' as described in Section 4.3.3.1:

$$r' = \mathcal{H}_{r,s}(B, g^k) + (g^a \bmod p) \bmod q$$

Similarly, each jury member uses the homomorphic properties of Shamir shares to compute a personal *share* $[s']$ from $[a]$ and their share $[k]$ of the redaction key:

$$[s'] = [a] - \text{SHA256}(B' || r') \cdot [k] \bmod q$$

Finally, the jury members recombine their shares $[s']$ so that, afterward, every jury member knows $(r', s') = \xi'$.

In conclusion, this distributed variant of the collision-finding algorithm can efficiently and securely realize redactions without disclosing secret redaction keys.

4.3.6.2 Updating Transactions and Blocks

Before a redaction jury can engage in the decentralized collision-finding algorithm we presented in Section 4.3.6.1, they have to modify the affected transaction and its enclosing block. We now present this process as well as its underlying rule set.

Replacing Transactions. Each jury member locally modifies transactions that shall be redacted depending on the applied insertion method. The insertion method is taken from the user's report (cf. Section 4.3.5) and, broadly, covers *intended* methods or any form of *address manipulation*. We now discuss how the jury members modify transactions from either category and how they deal with *other* insertion methods.

When content was inserted via an *intended method*, then it was either stored as an `OP_RETURN` payload or in the coinbase field (cf. Section 3.3.1). Both fields cannot be referenced by other transactions. Thus, the jury members can safely remove these fields as part of the redaction process without affecting the validation of other blocks.

When *address manipulation* has been used to store the content, the jury has to be more cautious, as this insertion method relies on manipulating spendable transaction outputs with associated coins. Hence, modifying such transaction outputs has potential side effects, as (a) coins can be permanently burned and (b) some outputs of a reported transaction might be spendable despite the manipulation or already spent. We mitigate these side effects by only *irreversibly obfuscating* content from address-manipulated transactions, i.e., the content cannot be recovered after the redaction, but any spendable output remains spendable. To this end, we cryptographically hash the mutable on-chain addresses a second time instead of removing them, which is in line with the previously proposed strategy for local content erasure by Florian et al. [FHBS19] (cf. Section 4.2.1) and works similarly to the hardened on-chain addresses we proposed in Section 4.2.5. Namely, when a user attempts to spend

an obfuscated output, full nodes can still validate that transaction by executing the additional hashing operation on the fly.

However, *other insertion methods*, such as using non-standard transactions or input stuffing, cannot be handled via simple modifications. Non-standard transactions can reliably be identified, but they can have highly diverse layouts as they are not bound to Bitcoin’s accepted standard payment patterns (cf. Section 2.3.3.3), and non-standard outputs are potentially spendable. As a result, finding proper side effect-free modifications to redact content from non-standard transactions would have to be attempted on a per-transaction basis. Moreover, redacting content from transaction inputs inherently breaks the transaction graph, as each non-coinbase input that is accepted into the blockchain has to satisfy the spending condition of a previous transaction’s output by definition. Removing such content thus creates a similar situation to that of local content erasure (cf. Section 4.2.1), where full nodes have to believe in good faith that the modified transaction has been properly validated before its redaction. RedactChain thus mitigates content insertion via non-standard transactions and input stuffing by thoroughly enforcing the stricter standardness tests also for those transactions that are included in a block (cf. Section 2.4.2) as well as P2SH redeem scripts (which should also follow the standard patterns according to BIP 16 [And12a]).

Updating Blocks. After updating a transaction, the redacting jury J_i updates the corresponding block (cf. Figure 4.7) to conclude the redaction. Each member computes an update entry describing the modification and appends it to the block’s update history (cf. Section 4.3.6.3). Each member also updates the Merkle root M and the history root R according to the changes. Then, the jury can jointly update the check value ξ using the secure collision computation (cf. Section 4.3.6.1). This way, the confirmation link H_P and the validity link $H_{V,i}$ of the block’s successor remain intact and assert that the modified block was previously confirmed and that it was altered by one of the responsible juries. Finally, the jury broadcasts the updated block, and other nodes accept the changes after verifying them.

4.3.6.3 Ensuring the Transparency of Redactions

Besides preserving the blockchain’s integrity via its CHF-based redactions, our redaction scheme remains transparent about its activities for all participants. In this section, we give an overview of the data structures used to realize this goal.

Update Entries. An *update entry* describes one action taken by a redaction jury and can either refer to a new modification (a redaction) or the reversal of another jury’s redaction (a restoration, cf. Section 4.3.7.2). Each entry is indexed by a block-level counter, which is incremented for each edit, and the index i of the redacting jury J_i . Each entry summarizes the block’s state before the edit as well as the reason for the edit based on the initial user report. The j -th edit of a block covers the block’s previous state by stating the edited transaction’s old identifier t_{j-1} , the old Merkle root M_{j-1} and history root R_{j-1} , and the old check value ξ_{j-1} . This way, other nodes can verify that the block was in a valid state before, reasons for redactions become

public, and keeping track of old transaction identifiers helps nodes to validate future transactions referencing edited transactions by their superseded identifiers.

Update History. RedactChain maintains a separate *update history* for each block. The update history contains a list of all update entries affecting the block and is tied to the block’s header via the history root R . The update history serves two purposes. First, nodes can comprehend and verify every past edit of the block. Second, also joining nodes can retrospectively learn about, and keep track of, any changes to transaction identifiers that occurred as a result of an edit. This way, joining nodes can still fully revalidate all past transactions, even if a transaction spent an output of a transaction that was only later edited by a redaction.

Redaction Log. In addition to per-block update histories, RedactChain also keeps a global *redaction log* of all activities so that juries can swiftly disseminate their modifications. Moreover, full nodes only accept those blockchain modifications that have a confirmed entry on the redaction log. The redaction log is a separate append-only ledger that is publicly readable but only extended by currently active juries via jointly created signatures. Juries publish their jury assembly block (cf. Section 4.3.7.1) as well as the update entries of their blockchain edits on the redaction log. Each update entry is additionally encapsulated by an envelope containing back links that confirm prior entries of the redaction log, a timestamp of the entry’s creation, and the jury’s signature. We use Nyberg-Rueppel signatures [NR95] to extend the redaction log due to their relation to the CHF scheme used by RedactChain. Namely, creating a signature also involves a random value and, thus, we can piggyback an additional instance of DKG onto the decentralized collision-finding algorithm (cf. Section 4.3.6.1) to draw this random value in a decentralized manner as well.

The redaction log further establishes consensus about the *approximate timing* of events among all nodes due to its coupling to the main blockchain. Each block references the redaction log’s current tip via H_{RL} (cf. Figure 4.7). Thereby, the block confirms that all entries on the redaction log existed at the time the block was mined. Even though dishonest miners may attempt to skew this timing by ignoring legitimate entries, honest miners will faithfully confirm the most recent entries to confirm all legitimate entries in a timely manner. Dishonest miners further cannot predate entries because the signature-based linking prevents inserting entries into an existing path. Finally, miners can merge redaction log forks and confirm the insertion time also of entries deliberately appended to old entries by confirming multiple tips at once. As we further elaborate in Section 4.3.7.2, nodes can use this approximate timing for coordinating whether to accept proposed modifications.

4.3.7 Coordinating Short-Lived Redaction Juries

The last main component of RedactChain’s design is its *coordination* between different and simultaneously active redaction juries (Step ④). In this section, we first present the election process to select and bootstrap the members of a new set of redaction juries (Section 4.3.7.1) and then detail how these juries coordinate their

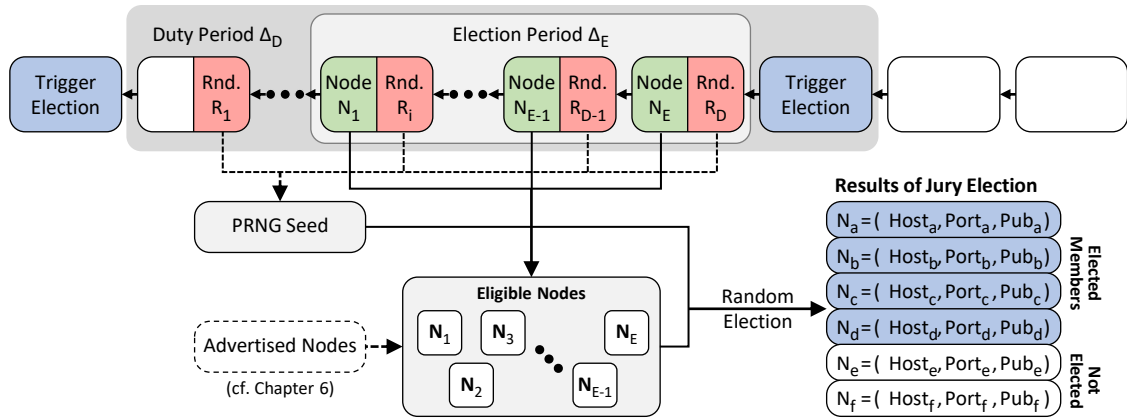


Figure 4.8 Jury election is triggered periodically when the current jury duty concludes, i.e., every Δ_D blocks. The local jury election selects n members from a pseudo-random shuffling of the last Δ_E successful miners. Optionally, further eligible nodes can be considered based on Sybil-resistant advertisements (cf. Chapter 6).

activities and exercise mutual oversight (Section 4.3.7.2). This step involves deliberately giving each redaction jury only control over a limited part of the blockchain for a limited time; we thus also outline how RedactChain can fall back onto the voting-based redaction scheme by Deuber et al. [DMT19] so that also unwanted content identified after a long time remains removable (Section 4.3.7.3).

4.3.7.1 Jury Election Process

Figure 4.8 shows the election process for each jury J_i consisting of n members. New juries are elected every Δ_D blocks from a pool of eligible nodes. By default, RedactChain considers all successful miners of the Δ_E most recent blocks eligible to be voted into a jury. Each miner includes its host name, port, and a public key in its blocks so that juries can bootstrap and users can create an index of how to reach active jury members to file a report. For increased diversity, RedactChain can also be extended using our Sybil-resistant bootstrapping service, AnonBoot, which we present in Chapter 6, to make also non-miners eligible to become jury members in a secure manner (cf. Section 6.6.3). After Δ_D blocks, each node locally creates an eligibility list from this data and shuffles this list based on a pseudo-random number generator (PRNG) that is seeded with randomness drawn from the last Δ_D blocks. RedactChain can be extended with dedicated randomness extractors [BCG15] to further reduce bias from blockchain-sampled randomness. We elect all i juries in parallel by repeatedly hashing the initial PRNG seed and reshuffling the eligibility list accordingly. The nodes then consider the n topmost nodes of the shuffled eligibility list as the elected members of J_i . The elected members also learn their exact position in J_i and how to connect to the other jury members. The juries can then bootstrap and create their jury assembly block. The members sign the assembly block; other nodes only accept assembly blocks with at least $x \geq 2n/3$ valid signatures. If a jury cannot announce the assembly block within a short time (e.g., < 10 blocks), the nodes elect a jury J_{m+1} in its place. Finally, the jury publishes the assembly block to the redaction log and starts accepting reports.

4.3.7.2 Coordination and Mutual Oversight

Concurrently active juries ensure that RedactChain remains actionable and accountable even if single juries misbehave. We now present how the redaction log and its approximate timing benefit the coordination of juries and enable a fair appeal process. Further, we briefly discuss conflicts stemming from concurrent redactions.

Approximate Timing for Coordination. Nodes mainly use the mining process to coordinate, e.g., new juries are elected every Δ_D blocks. Coupling the redaction log to the main blockchain (cf. Section 4.3.6.3) further enables an approximate timing for when edits happened. Namely, the nodes only consider edits when they were written to the redaction log and confirmed (directly or indirectly) on the main blockchain, which establishes the approximate timing of the edit used for coordination. The nodes use this timing to keep track of per-transaction *redaction timers* $\Delta_{R,t}$, i.e., the number of blocks since the last edit of transaction t . A transaction's redaction timer expires if it exceeds a threshold Δ_R , meaning that an edit to transaction t is only accepted as long as its redaction log entry is confirmed at the latest Δ_R blocks after the transaction was added to the blockchain. After Δ_R blocks, the redaction timer $\Delta_{R,t}$ elapses and full nodes will reject any edit proposed to the redaction log afterward. In this case, t can only be redacted via a fallback mechanism that consists of a slower public voting (cf. Section 4.3.7.3). However, each node resets the redaction timer $\Delta_{R,t}$ whenever a valid edit of t is confirmed on the redaction log before $\Delta_{R,t}$ expires. The reset gives another jury the chance to handle appeals by disagreeing users, as we discuss in the following.

Fair Appeal Process. Appeals are necessary to settle disputes stemming from manual jury votes or to revert unwarranted redactions. Our multi-jury approach enables users to request reverting a past decision before another jury. Users can ask a jury to restore a redacted transaction t by presenting a cached copy of the original state of t . This way, the jury can revert the redaction of t while still allowing all but appealing nodes to immediately remove or obfuscate t locally. If the appeal is successful, the nodes will reset $\Delta_{R,t}$, but keep track of which juries already edited t to reject future modification attempts from those juries. Hence, even strongly disputed transactions ultimately reach a final state, as their redaction timer will expire at some point due to no jury intending or being able to issue another edit. As we further elaborate in Section 4.3.8.2, keeping an odd number of active juries m ensures that a faithful jury will have control over the final decision as long as only $x < m/2$ juries are malicious. However, appeals can cause redaction timers to expire after the juries' duty ends. In such cases, juries may resolve pending disputes before dissolving.

Handling Conflicting Redactions. As juries are active concurrently, their modifications can provoke *conflicts* when they edit the same block. Nodes order the edits by their timestamps on the redaction log to prevent accidentally overwriting other edits. In the worst case, the jury losing this tiebreaker has to redo its edit. Since juries are intentionally small (e.g., tens of nodes), they can coordinate intended edits even before they are confirmed on the redaction log to lower the risk of collisions. Further, the most resource-intensive step of a redaction is using DKG to obtain a collision value (cf. Section 4.3.8.3), which is independent of the transaction to be

redacted. Unfortunately, a jury must not reuse its collision value after its updated check value is known (cf. Section 4.3.3.1), i.e., redoing DKG is required, but juries can delay applying a specific collision value until after the off-chain coordination.

4.3.7.3 Integration of Other Moderation Schemes

While RedactChain’s redaction process enables swift redactions without relying on fixed redactors, older transactions become immutable again and could potentially contain overseen content. To mitigate this risk and keep transactions redactable even if its redaction timer expired, RedactChain remains compatible with the voting-based redaction scheme by Deuber et al. [DMT19]. Both approaches use an immutable confirmation link as a fallback to validate a block’s initial state and a validity link that is invalidated as soon as the block is edited. Hence, RedactChain can be extended to offer a similar long-term voting process for older transactions. The fallback mode is triggered when a user proposes to modify a transaction after its redaction timer expired. In this case, all miners vote on-chain, independent of juries and redaction timers, and all nodes can observe the voting process. This approach introduces significant delays over RedactChain’s default redaction process but enables the removal of deeply engraved content. Similarly, the prevention strategies we presented in Section 4.2 can be deployed orthogonally to support RedactChain nodes and reduce the expected number of required blockchain modifications.

4.3.8 Evaluation

We now evaluate RedactChain’s effectiveness, security, and performance. We first argue that our redaction scheme can effectively handle all insertion methods and that its reduced immutability does not create invasive side effects (Section 4.3.8.1). We then discuss RedactChain’s resilience against adversaries seeking to control redaction juries (Section 4.3.8.2). Finally, we assess the performance of RedactChain by investigating the time required to execute redactions (Section 4.3.8.3) and the storage overhead created to establish transparency (Section 4.3.8.4).

4.3.8.1 Effectiveness and Non-Invasiveness of Redactions

We start our evaluation by arguing that RedactChain provides a holistic rule set for modifying permissionless blockchains, and by outlining how it protects the integrity of both the blockchain and the transaction graph despite potential modifications.

RedactChain deliberately determines the strategy to be applied by the redacting jury members based on the user’s initial report (cf. Section 4.3.5) to tailor the required modification to the applied content insertion mode (cf. Section 4.3.6.2). Namely, (a) small chunks of content inserted via intended methods (`OP_RETURN` or `coinbase`) are simply removed after a manual vote, (b) transactions reported for address manipulation are redacted by obfuscating their outputs instead of removing them, and (c) other insertion methods, i.e., non-standard transactions and input stuffing, are

prohibited by stricter validation rules. Furthermore, RedactChain keeps track of all past identifiers of a modified transaction, such that full nodes can still look up any referenced transaction despite potential modifications.

Deleting content inserted as an `OP_RETURN` payload or in the coinbase field can safely be done, as neither method has implications on the transaction graph. Thus, deleting those payloads cannot interfere with future spending attempts beyond the changing transaction identifier, as discussed above. Contrarily, removing potentially spendable outputs, e.g., address-manipulated P2PKH outputs, can alter the transaction graph. However, the obfuscation of spendable transaction outputs prevents breaking the transaction graph, as any spendable transaction output remains spendable after the obfuscation: Full nodes can still validate pending transactions even when they reference obfuscated outputs by replaying that obfuscation on the fly and, again, accounting for altered transaction identifiers. Since input scripts, e.g., for P2SH outputs, are inherently tied to the transaction graph’s integrity, RedactChain does not modify them, and the inclusion of content within those scripts must rather be prevented (cf. Section 4.2) or handled locally, e.g., via local erasure [FHBS19]. Finally, RedactChain prohibits non-standard transactions to reduce the complexity of executing redactions. If new payment patterns should receive wide acceptance and become standard patterns in the future, RedactChain’s rule set may be further extended to cope with these new patterns’ potential for content insertion.

Lastly, RedactChain’s compatibility with long-term voting-based redactions (cf. Section 4.3.7.3) ensures that also older transactions remain redactable in cases where unwanted blockchain content is identified only after all responsible redaction juries already dissolved. Thereby, RedactChain accepts the latency introduced by this fallback mechanism compared to the otherwise swift redaction by redaction juries to maintain full coverage of the blockchain regarding possibly required modifications.

Takeaway. RedactChain’s swift and transparent redactions maintain a high effectiveness to prevent content insertion via methods using non-standard scripts and provides a clear rule set to redact all other content without creating undesirable side effects. Overall, RedactChain extends upon the limited redactability achieved by related work, which often only considers provably unspendable transaction outputs.

4.3.8.2 Security Discussion: Adversary Resilience

We assess RedactChain’s security by considering a malicious adversary who intends to stall redactions or perform rogue modifications based on their share of nodes eligible for jury election and the other nodes’ verifiability of redactions.

An adversary can modify a transaction t if (a) they know one of the responsible juries’ secret redaction keys *and* (b) the redaction timer $\Delta_{R,t}$ has not yet expired. Redaction keys are generated using DKG (cf. Section 4.3.6.1); hence, the adversary can recombine a jury’s key and then compute collisions for the jury’s CHF to modify blocks if they control $x \geq n/2$ jury members (cf. Section 4.3.3.2). In this case, we say that the adversary successfully *infiltrated* a jury. Even though the adversary could technically execute redactions at will, they must still adhere to the rule set that

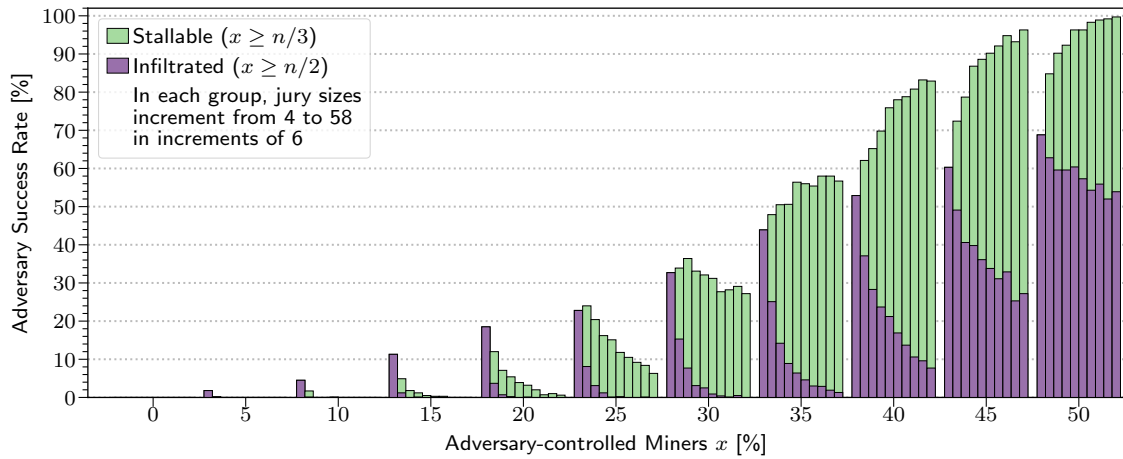


Figure 4.9 With growing control, a malicious adversary increases their chances to successfully stall or infiltrate a single jury (bars are not stacked).

defines valid modifications (cf. Section 4.3.6.2) and announce those modifications on the redaction log, as full nodes would reject the modification otherwise (cf. Section 4.3.6.3). Hence, the adversary can delete only `OP_RETURN` and coinbase payloads, but they must correctly obfuscate potentially spendable outputs, i.e., the adversary cannot alter the transaction graph even after infiltrating a jury. Furthermore, the full nodes detect attempts to modify a transaction more than once via the entries on the redaction log. If the adversary controls $n/3 \leq x < n/2$ jury members, they can only stall the final recombination step and prevent the honest members from executing redactions. While this strategy can delay a redaction, users can report the transaction to another, actionable jury (cf. Section 4.3.7.2) or ultimately fall back to a long-term public voting scheme (cf. Section 4.3.7.3). As long as the adversary controls only $x < n/3$ members, they cannot affect RedactChain’s operability.

Figure 4.9 shows the probability that an adversary can stall or infiltrate a single jury. We simulated an adversary who controls an increasing share of a total of 100 equal miners. We fix the duty period at $\Delta_D = 1000$ blocks (roughly one week of real-world time, considering Bitcoin’s target inter-block interval of 10 min) and the election period at $\Delta_E = 150$ blocks (roughly one day). We then elect 1000 juries of 4, 10, \dots , 58 members from the randomly chosen miners of the last Δ_E blocks based on a seed derived from the block identifiers of a randomly chosen sequence of $\Delta_D + 1$ blocks accepted on the blockchain of Bitcoin Core. This simulation shows that, while especially large juries only face a low infiltration risk from an adversary controlling up to 35% of the miners, there is a non-negligible risk that an adversary can stall a jury once controlling at least 25% of the miners.

We can further reduce this risk by relying on multiple, mutually overseeing juries. An odd number of juries ensures that an honest jury will have control over the final decision for at most $y < m/2$ infiltrated juries. Figure 4.10 gives the success rate of the adversary of gaining control over $y > m/2$ juries for $m = 1, 3, 5, 7, 9$ and a fixed, moderate (cf. Section 4.3.8.3) jury size of $n = 28$. While the adversary has a success rate of 15.1% to stall a single jury when controlling 25% of all miners, they have a lowered chance of only 3.8% of stalling redactions for $m = 9$ active juries. Notably,

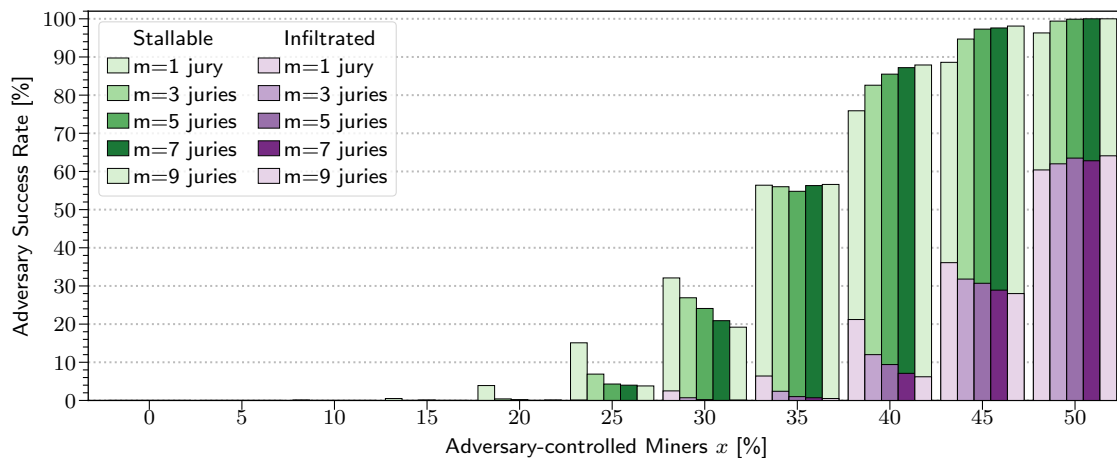


Figure 4.10 Relying on multiple, mutually overseeing juries improves RedactChain’s resilience against an adversary’s efforts. In the example above ($n = 28$), RedactChain remains resilient against an adversary controlling up to 25 % of all eligible miners (bars are not stacked).

this risk is lower than relying on a single jury of $n = 58$ members. Even though the adversary can stall redactions temporarily in rare cases, the network can still resort to public voting as a fallback (cf. Section 4.3.7.3).

Takeaway. Through its utilization of mutually overseeing juries, RedactChain remains robust against an adversary that controls up to $1/4$ of the overall mining power. While this threshold seems low, it is in line with practical results from considering selfish mining [ES14, SSZ17]. Further, our rule set for valid redactions as well as the option to ultimately fall back onto an on-chain voting process additionally thwart an adversary’s efforts to influence blockchain modifications.

4.3.8.3 Redaction Time

We now evaluate the time it takes a jury to redact a transaction, i.e., modify it and execute the decentralized collision-finding algorithm, to show that RedactChain can react swiftly to reported content. Furthermore, we assess RedactChain’s ability to scale to large network sizes.

Measurement Setup. We measure the redaction time for a single jury based on our Python prototype [MAP⁺22b], which uses the modules `aihttp` for communication and an adapted version of `python-bitcoinlib`, which we extended to handle RedactChain’s redactable blocks. All jury members run on the same server ($2 \times$ Intel Xeon Silver 4116, 196 GB RAM) and communicate over local TCP connections via RSA-signed messages. We use 2048-bit primes for our cryptographic primitives, i.e., whenever we use CHFs, Shamir’s secret sharing, or RSA. We create a simulated blockchain of redactable blocks without mining difficulty by implementing a pre-calculated transaction graph using the adapted `python-bitcoinlib` module. This blockchain consists of a genesis block for distributing initial funds and 1000 blocks with 1000 transactions each. Each block contains two redact-worthy transactions, one with 50 P2PKH outputs and one with an `OP_RETURN` output. From this

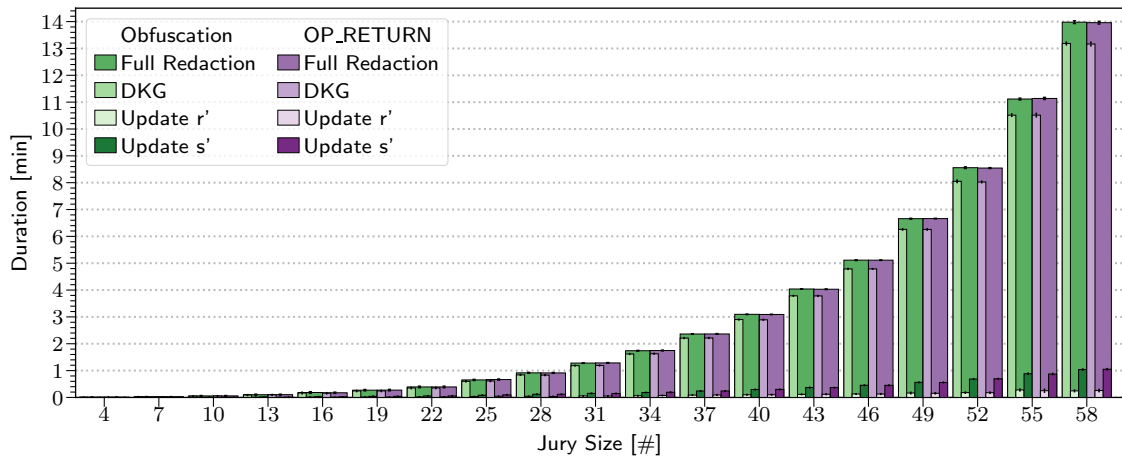


Figure 4.11 Even though the redaction times grow with the juries' size, mainly due to DKG, RedactChain remains capable of removing objectionable content in a timely manner.

blockchain, one fixed redaction jury of increasing size redacts both redact-worthy transactions from the first 30 blocks. We give the duration of the measured redaction steps by averaging the difference between the earliest jury member entering the phase and the last jury member concluding it for each redaction over the 30 blocks. We further give 99% confidence intervals for our measurements. Since the phases can overlap between jury members, the sum of phases can be larger than the overall redaction time.

Measurement Results. Figure 4.11 shows the time it takes a single jury of sizes 4, 7, ..., 58 to redact a transaction after accepting a user's report. We additionally highlight the main steps of the distributed collision computation, i.e., obtaining a collision value using DKG and computing the new check value components r' (local) and s' (involves Shamir recombination).

Our results indicate no substantial difference between obfuscating large transactions and redacting an `OP_RETURN` output. However, the redaction time increases super-linearly for larger jury sizes, mainly due to DKG. For instance, obfuscating a single large transaction takes a jury with $n = 58$ members 13.98 min, whereas a jury with $n = 28$ only needs 55.0s for a redaction, and that time is further reduced to 11.0s for a small jury with $n = 16$ members. Hence, relying on multiple but smaller juries can reduce the overhead of individual redactions without forfeiting RedactChain's resilience against adversaries (cf. Section 4.3.8.2). We have to derive secret-shared collision values using DKG, since knowledge of a full collision value allows retrieving the jury's secret redaction key (cf. Section 4.3.3.1). However, juries can pre-compute an appropriate number of DKG values after their assembly because the collision value is independent of the required blockchain modification. While this approach does not reduce the overall performance overhead of a redaction, it can vastly improve the jury's reaction time to user reports. Finally, these redaction times are independent of the total network size.

Takeaway. RedactChain can indeed swiftly react to the insertion of blockchain content and allows for timely redactions by keeping jury sizes moderate.

4.3.8.4 Overhead of Additional Information

RedactChain tracks additional information to keep redactions publicly verifiable. We now analyze the overhead created by tracking this metadata, i.e., our block structure compared to Bitcoin's, the jury assembly blocks, and the redaction log.

Block Header. Bitcoin blocks have a fixed-length 80 Byte-long header (cf. Section 2.2.2). In addition to this header's fields, RedactChain's block headers keep track of the original Merkle root M_0 (32 Byte), the $1 \leq i \leq m$ confirmations of the redaction log (1 Byte for the length, 32 Byte per link), the m CHF-based validity links (1 Byte for the length, 2048 bit = 256 Byte per link), the check value $\xi = (r, s)$ (2048 bit per component), and the history root R (32 Byte). Hence, the block header has a total size of 946 Byte for one jury and a worst-case size of 3250 Byte for nine juries confirming nine redaction log branches (i.e., one branch per jury). Since the majority of Bitcoin transactions have at most two outputs, and transactions consisting of one P2PKH input and two P2PKH outputs have an expected size of 225 Byte (cf. Section 4.2.5), even using nine juries would reduce the block capacity by only 15 transactions on average.

Redaction Log and History. Each modification has a corresponding update entry that is referenced in the update history of the modified block as well as the redaction log. As described in Section 4.3.6.3, an update entry consists of the per-block redaction counter (4 Byte), the index of the redacting jury (1 Byte), the block header's previous state as given by the old transaction identifier, Merkle tree root, and history root (32 Byte each), and the old check value (2×256 Byte) as well as a reason for the redaction. In total, an update entry has a size of 854 Byte, assuming that the reason given by the reporting user has a maximum length of 240 Byte (plus a 1 Byte-long length field). Each update entry is encapsulated on the redaction log and its envelope holds a timestamp (4 Byte), a Nyberg-Rueppel signature (256 Byte), and $1 \leq i \leq m$ confirmations of recent branches of the redaction log ($1 + i \cdot 32$ Byte). Hence, each entry on the redaction log has a total size that is between 1147 Byte and 1403 Byte. This overhead is significantly lower than recently achieved per-redaction overheads of 60 – 110 kB [LXY⁺22].

Jury Assembly Blocks. These blocks publish a new mining key (256 Byte) and hold signatures of $2n/3 < x \leq n$ jury members ($1 + x \cdot 256$ Byte). Jury assembly blocks are also part of the redaction log and thus hold i confirmation links ($1 + i \cdot 32$ Byte) as well. Hence, jury assembly blocks remain below 26.2 kB even for juries with 100 members, and they stay below 7.8 kB for moderate-sized juries of 28 members.

Takeaway. The overhead of redaction metadata remains low and is feasible to provide transparency and to coordinate nodes and redaction juries in RedactChain.

4.3.9 Summary and Future Work

RedactChain complements our initial approach of preventing that unwanted content is accepted into the blockchain (cf. Section 4.2) by providing a moderation framework

that allows for the simultaneously swift (**G1**) and transparent (**G2**) retrospective removal of blockchain content from permissionless blockchains (**Q1**, cf. Section 1.2). We achieve this desirable combination in a decentralized (**G3**) manner by periodically electing new distributed redaction juries that can only jointly execute modifications based on decentralized cryptographic building blocks, most notably distributed key generation (DKG) and a distributed variant of chameleon hash functions. Tracking all modifications on a dedicated redaction log does not only provide transparency but allows concurrently active redaction juries to coordinate (**P3**, cf. Section 1.1.2) and handle disputes while ensuring that such disputes are settled and each transaction’s state gets finalized except for a potential public long-term voting procedure among all miners as a fallback to tackle overseen content (**G4**). Furthermore, RedactChain defines a rule set for valid transactions and blockchain modifications (**G5**) that enables full nodes to validate such modifications and prevents malicious juries from overstepping their competencies. Finally, our evaluation shows that RedactChain is robust against an adversary who controls up to 1/4 of the total mining power (**P1**) and that jury sizes can be kept moderate to keep blockchain moderation scalable (**G6**) and overheads feasible (**P2**).

Overall, we demonstrated that also retrospective content moderation (**Q1**) can be achieved in the permissionless setting in a decentralized, swift, and transparent manner when carefully relaxing the requirement that all changes must be fully retrofittable to traditional blockchain systems (**Q2**).

However, we also identify further potential for improvements that we leave for future work regarding the *performance* of moderation frameworks based on redactable blockchains, the incorporation of *trained content moderators*, and the implications of blockchain modifications on *blockchain-backed services*.

First, while retaining feasible redaction times and storage overheads for moderate jury sizes, further performance improvements can increase the acceptance of deploying blockchain systems with retrospective moderation capabilities. Most notably, our utilization of DKG creates a substantial overhead that becomes a performance bottleneck for growing jury sizes. Furthermore, tracking blockchain modifications is a prerequisite for enabling transparent blockchain moderation, but this approach creates additional overhead on the blockchain. Future designs should investigate whether this overhead can be reduced on a global scale as well as potential security implications, e.g., when allowing only a subset of full nodes to keep track of all modifications similarly to pruning and non-pruning full nodes in Bitcoin today.

Second, RedactChain emphasizes the decentralization of moderation capabilities. However, content assessment and moderation is typically executed or supported by trained personnel with corresponding expertise [MSTP⁺22], such as the IWF [Int15a]. Hence, future work should analyze the potential to further incorporate this existing expertise, both to unburden other network participants and to improve the quality of moderation, without forfeiting the decentralization of data management in permissionless blockchain systems.

Finally, further implications of blockchain modifications should be taken into account. RedactChain focuses on keeping the transaction graph intact but does not

consider blockchain-backed applications. For example, deleting an `OP_RETURN` payload holding a critical control message for another application could have a serious impact on that application. Thus, further work is required to support the decision-making process of redaction juries, e.g., by providing specifications for application protocols that allow the jury to identify unintended deviations from that protocol. This way, jury members would be further guided to identify unwanted content in the protocol message, and they could refrain from redacting information that is crucial for an accepted blockchain-based service. In this regard, the format of user reports can be further improved to facilitate the automation of the decision-making process, i.e., reduce the number of required manual votes by a redaction jury.

4.4 Conclusion

In this chapter, we investigated how designs for permissionless blockchains can deal with the risks associated with unwanted content insertion by malicious actors (**P1**, cf. Section 1.1.2) we analyzed in Chapter 3. Overall, we have identified that content moderation must be an integral feature of blockchain-based data-management systems in the permissionless setting (**Q1**, cf. Section 1.2). We took two distinct approaches to realize urgently needed moderation capabilities.

On the one hand, we explored the design space for *prevention strategies* that involve only minimal changes to the designs of existing blockchains for handling digital currencies, such as Bitcoin, and thereby remain retrofittable more easily (**Q2**). Namely, we can mitigate content insertion by relying on only singular changes to the consensus rules, i.e., full nodes can take all decisions locally (**P3**).

On the other hand, we acknowledged that a full prevention of content insertion is not achievable and set out to also tackle the resulting need for enabling the *retrospective removal* of blockchain content. As any corresponding solution necessarily weakens the otherwise desirable immutability property of permissionless blockchains, retrospective moderation capabilities are not as easily deployable as prevention strategies. However, with RedactChain we showed that slightly relaxing this requirement opens up the possibility to realize decentralized, swift, and transparent reactions (**Q2**). RedactChain achieves this desirable combination of properties by relying on distributed cryptographic building blocks and by focusing on the coordination between the responsible nodes (**P3**).

Both approaches complement each other well, i.e., they can be combined in future blockchain designs: The retrospective removal of already engraved blockchain content allows rectifying oversights inevitably occurring when relying on prevention strategies. At the same time, these strategies can also reduce the expected number of blockchain modifications, which remain more costly than successfully preventing the insertion in the first place. Finally, we have shown that both approaches can be realized with only moderate overheads regarding the blockchain's size (**P2**). Nevertheless, large data volumes remain a pressing issue also for traditional blockchain systems. We investigate and address this issue in the next chapter.

5

Retrofitting Blockchains with Pruning Capabilities

In addition to the risks of non-financial content we discussed so far, blockchain-backed data-management systems face another serious challenge. Specifically, the append-only nature of traditional blockchain systems implies that blockchains grow continually as new data is stored, which creates increasingly massive data volumes that have to be replicated by all full nodes. In this chapter, we hence investigate the potential of permissionless blockchains to reduce the storage requirements for full nodes without compromising on the capability to bootstrap newly joining nodes. While we already discussed redactable blockchains as one method to retrospectively remove blockchain content in Section 4.3, those approaches are primarily concerned with modifying individual transactions or blocks. In the following, instead, we enable full nodes to forget old and obsoleted blockchain data via a novel and retrofittable block-pruning scheme, *CoinPrune* [MKP⁺20a, MKP⁺21].

We first motivate the need for retrofitting existing blockchains to mitigate the impacts of growing data volumes stored on permissionless blockchains (Section 5.1). Based on a thorough analysis of related work (Section 5.2), we then identify requirements for retrofittable block-pruning schemes (Section 5.3). Afterward, we give an overview of CoinPrune’s design (Section 5.4) before detailing its block-pruning scheme (Section 5.5), how it handles application-level data (Section 5.6), and how CoinPrune can be retrospectively deployed to Bitcoin on a technical level (Section 5.7). We then evaluate the security (Section 5.8) and performance (Section 5.9) of CoinPrune before concluding this chapter (Section 5.10).

5.1 Motivation

Cryptocurrencies undoubtedly benefited massively from the permissionless blockchains that serve as their underlying secure transaction ledger. As we have discussed in Section 2.1, especially the combination of decentralization, immutability, and public verifiability enables mutually distrusting peers to establish consensus about valid past events. As a result, blockchains have also increasingly been considered as backbones for the data management in other applications (cf. Section 3.4.1).

However, the combination of an increased usage of blockchain systems and their desired immutability aggravates the problem of ever-growing data volumes (**P2**, cf. Section 1.1.2): With growing popularity and increasingly complex use cases, more users submit more data to be stored on the blockchain, and full nodes have to keep a copy of the blockchain’s full history to enable public verifiability for newly joining nodes. The problem of growing data volumes, accompanied by increasing storage requirements for all full nodes, is exemplified by Bitcoin’s evolution over time: As of Jan 4, 2023, the blockchain of Bitcoin Core has a size of 416 GiB and grows by roughly 152 MiB every day [Blo11a]. Extrapolating this growth rate implies that Bitcoin’s blockchain could exceed a size of 500 GiB by Jul 24, 2024, i.e., most “casual” operators of full nodes would have to dedicate significant storage capabilities to just potentially serving joining nodes.

Resulting from already pressing storage requirements, operators of Bitcoin full nodes started to *prune* their blockchain copies in the past (cf. Section 2.5.1). Once a full node has fully validated a block (cf. Section 2.4.2) and updated its UTXO set based on the contained transactions (cf. Section 2.3.5), the block is only forwarded to newly joining nodes to aid their initial synchronization (cf. Section 2.5.4). Hence, maintaining a full blockchain copy has no inherent benefit for the full node and can be deleted after the initial synchronization process.

Even though pruning is rational from the perspective of the individual full node operator, running a pruning full node actively harms the overall health of the Bitcoin network: To root trust in Bitcoin’s current state, new nodes need to obtain and revalidate *all* blockchain data, including all data that became obsolete in the past, from independent sources. Pruning, however, reduces the availability of blockchain data that is required for maintaining public verifiability. Related work has proposed alternative blockchain designs featuring a built-in and global block-pruning process using *snapshot-based synchronization*, where joining nodes do not verify all blockchain data but instead rely on a recent snapshot of the blockchain’s state [Bru14, Poe16, CLO16, SM19]. While these efforts solve scalability challenges for new blockchain systems, their contributions do not readily carry over to existing and well-established blockchain systems, such as Bitcoin, as significant changes have to be adopted by a majority of nodes [Mor17]. Furthermore, the adoption of alternate blockchain designs depends on their distinction from already established systems; hence, features that could be implemented in Bitcoin are more unlikely to spur interest in new alternatives [SA21].

Consequently, there is a need and incentive to investigate how to *retrospectively extend established blockchain systems* with pruning capabilities (**Q2**, cf. Section 1.2).

Such retrofittable pruning schemes face fundamental design challenges. Namely, a viable scheme must mitigate a malicious actor’s intentions to tamper with a snapshot (**P1**), e.g., to introduce non-existing UTXOs, and the scheme must enable full nodes to coordinate pruning activities (**P3**) despite potential limitations of the original consensus protocol. Finally, block-pruning schemes need to be aware of non-financial blockchain content like we discussed in Chapter 3, i.e., they should limit the dissemination of objectionable content where possible (**Q1**) and support the benign data stored by higher-level applications (**Q3, P4**).

In the next section, we further discuss the potential benefits and the requirements for retrospectively deployable block-pruning schemes in greater detail.

5.1.1 Problem Analysis

In this section, we further discuss the need for retrofitting pruning capabilities to established blockchain systems and identify challenges for their deployment. To this end, we use Bitcoin as our working example for our following analysis and contributions. We begin by briefly recapitulating Bitcoin’s initial synchronization process (Section 5.1.1.1). Afterward, we highlight challenges regarding the operation of blockchain systems that are aggravated by long-term utilization and increased data volumes (Section 5.1.1.2). Finally, we outline challenges for retrofitting non-trivial changes to up-and-running permissionless blockchain systems (Section 5.1.1.3).

5.1.1.1 Recapitulation of Bitcoin’s Initial Synchronization Process

When first joining the Bitcoin network, new nodes start an initial synchronization process where they obtain and revalidate all blocks that were previously accepted into the blockchain (cf. Section 2.5.4). Namely, the joining node first chooses its neighbors, obtains the *headerchain*, i.e., the chain of all accepted block headers, from one neighbor afterward, and finally asks all neighbors for the full blocks.

However, the ultimate goal of this process boils down to letting the joining node derive the exact same UTXO set (cf. Section 2.3.5) as the fully synchronized full nodes. The joining node locally revalidates all obtained information (cf. Section 2.4.2) and applies each valid block’s transactions to its UTXO set. After catching up with the other full nodes, the initial synchronization is concluded and the node can start validating pending transactions solely based on its derived UTXO set (cf. Section 2.4.2). Hence, the UTXO set represents a full node’s relevant *state* required for reaching consensus with the other full nodes.

In principle, joining nodes could skip obtaining the full blockchain history if they were provided with an up-to-date UTXO set instead. However, joining nodes have no means to trust that any provided UTXO set correctly reflects the blockchain’s full history since the genesis block. Hence, the joining nodes must currently resort to obtaining and revalidating all blockchain data.

In the following, we discuss the impacts created by this performance-wise non-optimal synchronization process, with a focus on long-term blockchain utilization.

5.1.1.2 Impact of Long-Term Blockchain Utilization

Blockchains continuously grow in size by design due to their append-only nature. Thus, they will eventually reach prohibitive sizes. With a size of 416 GiB [Blo11a], the blockchain of Bitcoin Core has already reached critical volumes. Moreover, its growth rate of 152 MiB/d marks a worrying trend that 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, considering our results from Chapter 3, adding unwanted blockchain data may inflate the number of theoretically spendable UTXOs that will effectively never be spent, i.e., the UTXO set might be *polluted* with unwanted data over time.

Storage Requirements. To retain consensus within the network, Bitcoin requires that enough independent nodes maintain a full blockchain copy to bootstrap joining nodes. However, storing hundreds of gigabytes of past blockchain data is irrational for node operators and prohibitive for storage-constrained devices. Hence, 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 first obtain its 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., disseminating pending transactions or newly mined blocks.

Processing Costs. In addition to downloading the blockchain, joining nodes also need to verify its integrity and locally replay every single transaction to derive an up-to-date UTXO set. This process requires excessive amounts of computation power [Bit15d, Gen19, Lop20]. 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 processing costs combined cause prolonged synchronization times. For instance, benchmarks using powerful clients report synchronization times of 5 – 8 h between 2018 and 2020 [Gen19, Lop20]. Moreover, literature already highlighted this issue in 2016 when four days were required to bootstrap Amazon EC2 nodes [CDE⁺16]. 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, without dedicated redaction functionality (cf. Section 4.3), also unwanted blockchain content cannot be removed retrospectively. Since address-manipulated transaction outputs resemble spendable outputs and cannot be reliably detected as such (cf. Section 4.1.1.1), those transaction outputs are also included in the full nodes' UTXO sets. These transaction outputs are effectively unspendable and thus expected to remain in

the UTXO set indefinitely. On the one hand, such outputs bloat the UTXO set, which implies further scalability issues during the validation of pending transactions (cf. Section 2.4.2), unnecessarily. On the other hand, objectionable content added via address manipulation is also present in the UTXO set and not only the full blockchain copy. Hence, we say that unwanted blockchain content can also *pollute* the blockchain and the UTXO sets maintained by the full nodes (cf. Section 3.4.2.1).

Summary. Ever-growing blockchains imply strong scalability issues for both joining and established nodes, and the presence of illicit content presents 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. Hence, mitigating the negative impacts of long-term blockchain utilization is mandatory to ensure that permissionless blockchain systems remain operable.

In the following, we briefly outline the challenges hindering the retrospective deployment of corresponding measures.

5.1.1.3 Challenges for Retrofitting Consensus Updates

Previous changes to the consensus protocols of permissionless blockchains have shown that their deployment faces significant challenges. A prominent example in this regard is the intense discussion about the maximum block size in Bitcoin, which ultimately led to the deployment of Segregated Witnesses (SegWit) in Bitcoin Core (cf. Section 2.3.4). As we have discussed in Section 2.4.3.2, intentional changes to the consensus protocol are typically executed either via *hard forks* or *soft forks*. While hard forks introduce breaking changes, e.g., altered block structures, soft forks aim to remain backward-compatible with full nodes following older consensus rules, i.e., they only restrict the set of valid blocks. However, both types of forks can incur *permanent* splits of the blockchain, depending on whether the majority of nodes accept or reject the proposed changes [ZSJ⁺19]. Hence, either fork type bears risks when used to introduce new functionality in established blockchain systems.

Recently, a novel form of forks, *velvet forks*, has been proposed to address these kinds of deployability issues and allow for a *gradual deployment* of new features [KMZ20, ZSJ⁺19]. The key idea of a velvet fork is to introduce a new feature in a way that legacy full nodes are not affected by the rule changes. Velvet forks maintain two crucial properties to achieve the desired gradual deployability [ZSJ⁺19]: First, blocks are only updated in a way that they remain valid also to legacy full nodes, i.e., those nodes remain oblivious of the rule changes. Second, upgraded full nodes do not reject legacy blocks in order to maintain compatibility with legacy full nodes. Velvet forks can be implemented, for instance, using the coinbase field of each block [ZSJ⁺19], as it provides an intended way to insert small amounts of miner-defined data (cf. Section 3.3.1) and current full nodes ignore this payload.

In conclusion, retrofitting permissionless blockchains with additional features proves challenging, as the process bears the risk of creating permanent blockchain splits. However, velvet forks recently emerged as a promising middle ground between hard forks and soft forks to allow for gradually deploying such extensions.

5.1.2 Contributions

To address the pressing issue of ever-growing data volumes in permissionless blockchains (**P2**), we investigate the potential for retrofitting existing systems with block-pruning capabilities (**Q2**). To this end, we propose *CoinPrune*, our block-pruning scheme that is fully compatible with Bitcoin and that can be adopted immediately by any subset of full nodes without changing Bitcoin’s consensus rules due to its deployability via a velvet fork. At its core, CoinPrune provides a *trustworthy* (**P1**) 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 only need the small snapshots to synchronize, and therefore established full nodes can prune obsolete blocks *without* impeding the overall network health.

Overall, we present the following contributions toward relieving the storage pressure inflicted by ever-growing permissionless blockchains in this chapter:

- 1) We *comprehensively survey* existing approaches to decrease the storage requirements and synchronization times of blockchain systems, concluding that they are either inefficient, insecure, or not deployable to established systems.
- 2) We present CoinPrune, our snapshot-based pruning scheme that is fully compatible with Bitcoin. CoinPrune establishes trust against malicious interference (**P1**) because CoinPrune-supporting miners *periodically and independently reaffirm* the snapshots’ correctness by cryptographically tying them to the blockchain in the coinbase fields of their blocks. This way, CoinPrune-supporting miners and full nodes can coordinate their mutual understanding of each snapshot’s correctness (**P3**). Thus, assuming sufficient support for CoinPrune, joining nodes may trust the honest network majority to verify snapshots before relying on them. As legacy full nodes ignore reaffirmations instead of rejecting them, CoinPrune can be deployed mid-operation via a velvet fork [KMZ20, ZSJ⁺19] (**Q2**).
- 3) CoinPrune further renders objectionable content in the UTXO set (and thus its snapshots) harmless (**Q1**) by *obfuscating* the recorded values, similarly to the obfuscation scheme we applied for RedactChain (cf. Section 4.3.6.2). This way, content that would be obtainable from the UTXO set is made inaccessible without hindering the validation process that relies on the UTXO set (cf. Section 2.4.2). Simultaneously, CoinPrune preserves short chunks of non-financial data from `OP_RETURN` transaction outputs, which are essential for higher-level applications (**Q3, P4**), as we discussed in Section 3.4.1, in an additional *application data storage* that can be distributed alongside the snapshots.
- 4) Our evaluation shows that CoinPrune reduces disk space utilization for full nodes of Bitcoin Core by 87% while still allowing new nodes to synchronize. Pruning further 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 relative drops to be expected for 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.

- 5) We provide the source code for our prototypic implementation of CoinPrune that is based on Bitcoin Core v0.17.1 as open source under the MIT license [MKP⁺20b].

With these contributions, we emphasize that even long-running blockchain systems can be retroactively extended to cope with increasing data volumes and can therefore serve as a long-term backbone for data-management tasks.

5.2 Survey of Related Work

Learning from the scalability issues we summarized in our problem analysis (cf. Section 5.1.1), blockchain developers tackled the identified challenges from different perspectives. In this section, we survey measures to retrospectively extend established systems as well as alternative blockchain designs that focus on reducing storage requirements and improving the bootstrapping of new full nodes prior to CoinPrune.

5.2.1 Survey Criteria

Before presenting the results of our survey, we outline the criteria according to which we reviewed previous measures affecting storage requirements of permissionless blockchains. Overall, 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 permissionless blockchains, such as Bitcoin.

We assess *scalability improvements* by considering processing, traffic, and storage improvements separately. From this, we further 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 performance benchmarks that would allow for a quantitative comparison. Furthermore, we discuss to which extent the improvements impact their base systems' *security*. We also consider the approaches' *impact on the overall network* in terms of the network health, i.e., the overall network's dependency on especially altruistic nodes, and the potential overhead the approaches may introduce for already synchronized nodes. Moreover, we assess how the proposed schemes may impact the *blockchain queryability*, e.g., the capability of querying now-obsolete transactions or `OP_RETURN` payloads. Finally, we survey the schemes' *compatibility* with already deployed blockchain systems, with a special focus on retroactive deployability to Bitcoin. We summarize our results in Table 5.1.

5.2.2 Retrospective Changes to Established Blockchain Systems

With the increased popularity of blockchain-based cryptocurrencies, their respective developers were already forced to tackle the accompanying scalability issues. In our

Name of Approach	Category of Approach	Reduce Processing	Reduce Traffic	Reduce Storage	Synchronization Time	Maintain Security	Network Health	Server Burden	Query Completeness	Compatibility with Bitcoin
Hot Wallets [Bit12a]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	○	○	●	○	●
Light Nodes [Bit18d]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	●	○	●	○	●
"Ultimate Compression" [Rei12]	Trust Delegation	○ / ●	○ / ●	○ / ●	○ / ●	○	●	○	●	●
DB Improvements [Bit13a, Bit17b]	Data Management	●	○	○	●	●	●	●	●	●
Index Pruning [Bit13a]	Data Management	●	○	●	●	●	●	●	●	●
Headers-first Download [Bit15a]	Data Management	○	○	○	●	●	●	●	●	●
Assume-valid Blocks [Nak10, Bit17a]	Skip Verification	●	○	○	●	●	●	●	●	●
Block Pruning [Bit15b]	Simple Block Pruning	○	○	●	○	●	○	●	●	●
Ethereum Fast Sync [Szi15]	State-based	●	○	○	●	●	●	●	●	○
AssumeUTXO [O'B19]	State-based	○	○	○	●	●	●	●	●	●
Selective Pruning [PSB18]	Simple Block Pruning	●	●	●	●	○	●	○	●	○
Rollerchain [CLO16]	State-based	●	●	●	●	●	●	●	●	○
Marsalek et al. [MZFZ19]	State-based	○ / ●	○ / ●	○ / ●	○ / ●	●	●	●	●	○
Mini Blockchain Scheme [Bru14]	Balance-based	●	●	●	●	●	●	●	○	○
Mimblewimble [Poe16]	Balance-based	○	●	●	○	●	●	●	○	○
Pascal [SM19]	Balance-based	●	●	●	●	●	○	●	○	○
Vault [LSGZ19]	Balance-based	●	●	●	●	●	●	●	●	○
FlyClient [BK LZ20]	Commitment-based	○ / ●	○ / ●	○ / ●	○ / ●	●	○	●	○	○
TICK [ZYH+22]	Commitment-based	○ / ●	○ / ●	○ / ●	○ / ●	●	○	●	○	○
Minichain [CW20]	Commitment-based	●	●	●	●	●	●	●	○	○
CoinPrune (our approach)	State-based	●	●	●	●	●	●*	●*	●	●

○ / ●: Distinction Full Nodes / Light Nodes

*: Dependent on honest majority among adopters (cf. Section 5.8.2)

Table 5.1 Previous improvements tackling Bitcoin's scalability issues regarding its storage requirements only achieved limited effectiveness, whereas the approaches implemented by alternative blockchain designs cannot be easily integrated into long-running blockchain systems retroactively. Our proposed block-pruning scheme, CoinPrune, carries over these recent achievements to Bitcoin by enabling a gradual deployment via a velvet fork.

survey, we first focus on corresponding measures for already established blockchain systems that are either taken locally by the users or on a network-wide scale by the systems' developers. Our discussion is based on the reference implementations (Bitcoin Core and Ethereum's `geth`, respectively) where appropriate. Overall, we identify retrospectively deployed approaches that are based on *trust delegation*, *skipping verification steps*, *improving data management* with the special case of *block pruning*, and *state-based synchronization* in our survey.

Trust Delegation. Users resort to relying 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* [Bit12a], users essentially outsource all fund management to a trusted third party, enabling the service provider to issue transactions on their behalf. *Light nodes* [Bit18d] (cf. Section 2.5.1) similarly outsource validation tasks to other full nodes, but they manage their wallet locally using simple payment verification (SPV) (cf. Section 2.2.3). These actively used approaches vastly improve the performance as perceived by Bitcoin users and only put a negligible burden on the full nodes. However, using trust delegation, Bitcoin users only seize the resources of other full nodes and do not contribute positively to the overall network operation, as they rather rely heavily on a backbone network comprised of proper full nodes for both trust and relevant information. The never-deployed Ultimate Compression scheme [Rei12] aimed at bootstrapping light nodes with an up-to-date UTXO set, but this approach 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. Moreover, this scheme requires an additional blockchain to establish trust in the transmitted UTXO set.

Improving Data Management. Increasing blockchain sizes necessitate that all full nodes optimize their local data management, both for looking up relevant information and bootstrapping joining nodes efficiently. To this end, Bitcoin Core has historically changed its *underlying database* system [Bit13a], including the internal layout of its UTXO set [Bit17b], to facilitate faster processing. Furthermore, full nodes maintain a *transaction index* to look up relevant transactions faster, and they started to *locally prune* obsolete entries from this index by default [Bit13a]. While the nodes still persist the full raw blockchain data, such obsolete data is not queryable anymore. Network-related optimizations mainly encompass the revised *header-first download* [Bit15a] we discussed in Section 2.5.4. This change has been deployed with only minor and local upgrading incompatibilities [Bit15a]. Full nodes can further *limit their block-serving bandwidth* [Bit16a] and relay *compact representations* of newly mined blocks, which reduces redundancies that would otherwise occur due to rebroadcasting known-but-pending transactions [Bit16b]. However, despite these revisions to the dissemination of blockchain data via header-first download, full nodes are still required to obtain and process all data during their initial synchronization.

Skipping Verification. Early on, Bitcoin Core introduced functionality that allowed synchronizing nodes to avoid fully revalidating very old transactions. Using hard-coded *checkpoint blocks* at first [Nak10], Bitcoin Core later started using configurable *assumed-valid blocks* [Bit17a]. The rationale behind this relaxation is that any older transaction that was invalid would have been rejected instead of being confirmed

continually. Thus, older transactions that have accumulated sufficiently many confirmations are believed to be correct. By skipping the full validation of assumed-valid blocks, synchronizing nodes can omit the costly signature verification of large portions of the blockchain at negligible security risks. However, skipping verification also does not unburden joining nodes from downloading the complete blockchain, as they still have to replay all older transactions to derive an up-to-date UTXO set.

Simple Block Pruning. A more invasive means of reacting to increasing storage requirements is running a pruning full node, which *completely prunes the raw blockchain data* [Bit15b] after concluding the *full* initial synchronization process. Pruning full nodes deliberately forget about all obsolete blockchain data at the cost of its queryability. In contrast to pruning the transaction index as discussed above, running a pruning full node is detrimental to the network health, as the pruning nodes are incapable of bootstrapping new nodes.

State-based Synchronization. While Bitcoin maintains a focus on financial transactions, other blockchain systems, such as Ethereum [But13, Woo14] are based on smart contracts. Naturally, smart contract-based blockchain systems have more complex state layouts, because those systems' full nodes need to keep track of every smart contract's state. Ethereum uses *Fast Sync* [Szi15] to improve the initial synchronization process for full nodes. Using Fast Sync, joining nodes only download a recent state and can avoid having to replay all past transactions. However, Ethereum still values the queryability of older data, and joining nodes still download older blocks, albeit without processing them during the initial synchronization. Ethereum full nodes can trust Fast Sync since every Ethereum block includes a cryptographic link to the current state before applying the block's transactions [But13, Woo14]. Hence, and in contrast to Bitcoin, Ethereum's full nodes explicitly confirm the state's correctness when validating a block, and joining nodes can verify the correctness of their obtained state directly via Ethereum's blockchain. Conversely, Fast Sync is not immediately portable to other blockchain systems that lack these header fields, such as Bitcoin. AssumeUTXO [O'B19] aims for a similar extension for Bitcoin by extending upon its assume-valid blocks. Joining nodes bootstrap using a relatively recent UTXO set to be operable early on, but also perform a full synchronization in the background to retain full queryability; i.e., synchronization times are reduced, but processing, traffic, and storage requirements are not. Moreover, 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 blockchains from different angles. However, all approaches have at least one severe drawback, as they suffer from limited efficiency or questionable security properties, or they are detrimental to the network health or not readily portable to already established systems.

5.2.3 Alternative Blockchain Designs with Pruning Capabilities

The lacking efficiency of post-deployment measures motivated various *alternative blockchain designs* that promise further scalability improvements. In the following,

we identify and discuss alternative designs that pose refinements of *simple block-pruning schemes* as well as designs proposing *state-based* synchronization or keeping track of *account balances* instead of the full transaction history, and, finally, *commitment-based* designs that have the goal of further reducing or even removing the need for locally maintaining the current state by each full node.

Simple Block Pruning. Palm et al. [PSB18] present a distributed block-pruning scheme for established nodes in *permissioned* blockchains, which are 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 transactions declared irrelevant in a way that other nodes can still retrieve all data still considered relevant. As this pruning scheme focuses on permissioned blockchains and requires the definition of a dedicated initiator, the approach is inapplicable to the permissionless setting, where parties are not authenticated and may not be fully trusted.

State-based Synchronization. Similar to Ethereum’s Fast Sync, and inheriting its advantages and disadvantages, Rollerchain [CLO16] 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. [MZFZ19] 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. In Section 4.2.1, we have already briefly discussed a special class of state-based pruning schemes that mitigate unwanted content insertion by redefining what information is kept as the system’s state [Bru14, Poe16, SM19, LSGZ19]. However, the primary intention of these proposals is simplifying the state layout to allow for more effective pruning. Typically, these schemes only keep track of existing accounts and their balances. The mini blockchain scheme [Bru14] replaces Bitcoin’s UTXO set with an account tree that is 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 explicitly guaranteeing the availability of all required data to rewind the account tree accordingly. Mumblewimble [Poe16] follows a similar approach but emphasizes confidential transactions at the cost of synchronization performance, as joining nodes have to obtain and verify rangeproofs for unspent funds [Poe16]. Through their balance-based approach, both schemes limit the expressiveness of transactions compared to Bitcoin’s scripting capabilities. To overcome this limitation, Pascal [SM19] 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 [LSGZ19] builds upon Algorand [CM19] 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, e.g., Bitcoin’s Merkle

tree root (cf. Section 2.2.3), and make state data queryable this way [BKLZ20, ZYH⁺22, CW20]. FlyClient [BKLZ20] reduces the overhead for light nodes relying on SPV by relieving them from obtaining the full headerchain. FlyClient achieves this goal by maintaining a Merkle mountain range [Tod12] over the headerchain. A Merkle mountain range is an append-only variant of a Merkle tree that allows for efficient membership tests [BKLZ20]. In FlyClient, a light node uses this commitment to probabilistically challenge full nodes about the validity of their headerchain before accepting their blocks and continuing with SPV [BKLZ20]. TICK [ZYH⁺22] encodes the UTXO set as an AVL hash tree that commits to the UTXO set in a way that (a) modifications to the UTXO set can efficiently be reflected based on an existing AVL hash tree and (b) light nodes can efficiently verify the presence or absence of any specific UTXO based on an up-to-date AVL hash tree [ZYH⁺22]. Both approaches assume that full nodes store the full UTXO set, but they can provide light nodes with a succinct proof to convince them of their honesty. Contrarily, MiniChain [CW20] aims for a *stateless blockchain* that overcomes the need for storing the full UTXO set for verification purposes. Based on prior work by Boneh et al. [BBF19], MiniChain uses RSA-based accumulators. Similarly to Merkle trees, these accumulators allow for summarizing the state without having to store it explicitly. MiniChain then requires users to provide an additional proof alongside their transactions to convince validating nodes that the transaction only spends existing coins that have not previously been spent [CW20]. However, like balance-based schemes, these commitments are not retrospectively deployable to established systems.

Takeaway. Alternative blockchain designs have shown that incorporating cryptographic ties to recent state objects is promising to establish trust in state-based synchronization processes. However, extending established systems accordingly results in hard forks, which hinders the deployment of otherwise desirable features.

5.3 Design Goals

Traditional permissionless blockchains require a backbone of full nodes to individually maintain a *full local blockchain copy* (cf. Section 2.4.2). As we have discussed in Section 5.1.1.2, this initial design becomes massively burdening for these nodes as well as joining nodes; as a result, multiple approaches to fully pruning obsolete data have been proposed (cf. Section 5.2.3). However, none of these approaches can be adapted to directly provide similar optimizations for established systems, such as Bitcoin, without creating major incompatibilities. Furthermore, and to the best of our knowledge, no current block-pruning scheme considers data semantics, i.e., UTXO set pollution or the impact of pruning on higher-level application data. Accordingly, we identify the following requirements and design goals for block-pruning schemes that aim for providing *fully compatible* extensions for already established blockchain systems while *maintaining established security levels* and *respecting data semantics* to account for an advanced blockchain-backed data management:

(G1) Scalability. To be effective, block-pruning schemes must provide improvements for *all* metrics we identified in Section 5.1.1.2, 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 legacy nodes and updated nodes remain capable of reaching consensus about the system’s state. Specifically, any full 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, especially in the permissionless setting, block-pruning schemes must ensure that joining nodes remain capable of verifying the correctness of the data they process during the initial synchronization even in the presence of adversaries. Hence, 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 5.2.3), all changes should be retrofittable to existing blockchains, especially Bitcoin. Preferably, the scheme is opt-in, as is achievable via velvet forks (cf. Section 5.1.1.3).

(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 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` outputs.

With these requirements and design goals defined, we present our proposed and Bitcoin-compatible block-pruning scheme in the following.

5.4 CoinPrune Overview

To address the challenges stemming from ever-growing blockchain sizes, we present *CoinPrune*, our secure, snapshot-based block-pruning scheme that is gradually deployable without protocol-breaking changes to established permissionless blockchain systems, e.g., to Bitcoin. In this section, we first give an overview of *CoinPrune*’s design before presenting its block-pruning scheme, the way it handles additional data semantics, and its retroactive integration into Bitcoin in the next sections.

CoinPrune is designed to transfer the scalability improvements achieved by alternative blockchain designs (cf. Section 5.2.3) to Bitcoin while keeping compatibility a priority. We now describe how *CoinPrune nodes*, i.e., Bitcoin full nodes that additionally support *CoinPrune*, jointly maintain recent snapshots via the blockchain and how new nodes can bootstrap securely via those snapshots. Finally, we outline the benefits for the whole network that are achieved by *CoinPrune*.

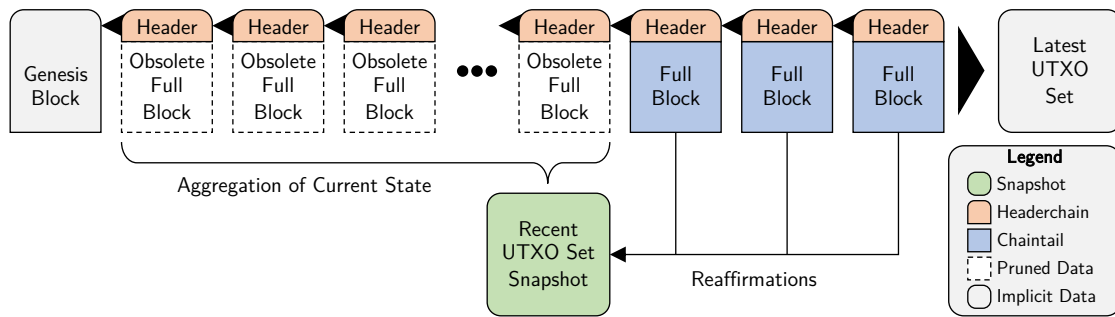


Figure 5.1 Instead of obtaining and revalidating all blocks, CoinPrune nodes can securely synchronize using a recent snapshot of the UTXO set due to reaffirmations by the miners.

Snapshot Maintenance. CoinPrune nodes periodically create *snapshots* of their UTXO set. These snapshots are created to help bootstrap newly joining nodes. Instead of obtaining a full blockchain copy, joining nodes request a recent snapshot to improve their initial synchronization and lower its storage, bandwidth, and processing requirements (**G1**). Snapshots are a well-ordered representation of the UTXO set that are used to facilitate the bootstrapping process (**G2**) and they are cryptographically tied to their corresponding block height (cf. Section 2.2.1). To prevent malicious nodes from distributing invalid snapshots, e.g., in an attempt to multiply their funds by including non-existent UTXOs, CoinPrune requires snapshots to be *publicly announced* on the blockchain by referencing a cryptographic *identifier* of each snapshot on-chain. CoinPrune miners place these announcements in their blocks' coinbase fields (cf. Section 3.3.1). Using the coinbase fields is essential to maintain compatibility with Bitcoin (**G4**), as current full nodes ignore these payloads when validating blocks. Hence, CoinPrune miners can include links to recent snapshots in their blocks without any change to the consensus rules. Other CoinPrune miners independently do the same, which causes miners referencing the same snapshot to *mutually reaffirm* its validity. This approach creates positive-only feedback, i.e., wrong snapshots are not rejected but tolerated and outpaced by the reaffirmations of valid snapshots 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 based on the blockchain-linked snapshots, as shown in Figure 5.1: First, the node obtains a *recent snapshot* either from its neighbors or through a third party (cf. Section 5.8.1). When acquiring the snapshot from its neighbors, the joining node downloads the snapshot advocated by the majority of the neighbors. Second, the node obtains the *headerchain*, just as it would do using headers-first download (cf. Section 5.2.2), to learn about the blockchain branch containing the most PoW. 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 accepted after the snapshot was created and (b) inspect the full blocks for reaffirmations of the snapshot used for synchronization. If the joining node observes sufficiently many reaffirmations, it accepts the snapshot and concludes the initial synchronization. Otherwise, the node discards the insecure snapshot and retries after reconnecting to new neighbors.

Global Block Pruning. Since joining nodes can securely bootstrap from the head-erchain, snapshot, and 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. 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 (a) that the UTXO set may contain address-manipulated UTXOs that can be unspendable or encode objectionable content and (b) that small chunks of application data are stored in `OP_RETURN` transaction outputs, which are desirable to preserve but inherently prunable. To account for these higher-level data semantics (**G5**), CoinPrune nodes can *locally obfuscate* most manipulable on-chain addresses without impeding the nodes' capability of validating pending transactions, and they maintain an additional *application data storage*, which contains all `OP_RETURN` payloads and their context.

In the following, we present CoinPrune's block-pruning scheme, its handling of data semantics, and its retrospective deployability to Bitcoin in more detail.

5.5 Retrofittable Block Pruning

We begin our in-depth presentation of CoinPrune by detailing its most crucial component, its retrofittable block-pruning scheme for permissionless blockchains.

First, we discuss how CoinPrune nodes adapt their local data management to facilitate the pruning process (Section 5.5.1). Then, we present how the nodes coordinate to establish consensus about the validity of snapshots (Section 5.5.2). Finally, we detail how joining nodes can bootstrap efficiently using CoinPrune (Section 5.5.3).

5.5.1 Data Management

To seize the improvements promised by switching to a snapshot-based initial synchronization process, CoinPrune nodes have to adapt their local data management. In the following, we thus present how CoinPrune nodes derive snapshots from recent UTXO sets and what data they still have to maintain locally when using CoinPrune.

Snapshot Layout and Creation

A CoinPrune snapshot is a deterministic serialization of the UTXO set as derived from processing all transactions recorded on the whole blockchain up to a specific block height. Hence, each snapshot aggregates the whole transaction history up to that block height. CoinPrune reinforces this tight link between a snapshot and the blockchain via a *two-way binding* where snapshots reference their corresponding block height and the snapshot is reaffirmed on the blockchain in subsequent blocks.

A snapshot consists of a simple header and a body, and each snapshot has an associated cryptographic identifier to be referenced on the blockchain. The *snapshot*

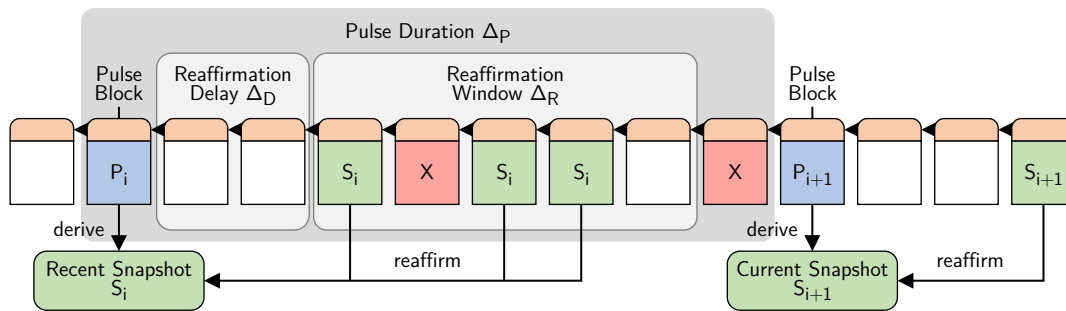


Figure 5.2 CoinPrune nodes coordinate based on additional consensus rules: Certain blocks are interpreted as pulse blocks, which are the basis for creating a new snapshot as well as the start of a new pulse of duration Δ_P . During a reaffirmation window of Δ_R blocks mined after a short reaffirmation delay of Δ_D blocks after the pulse block, miners indicate the correctness of the last snapshot in the blocks they mine. Any invalid or delayed reaffirmations are ignored.

header holds the snapshot's associated block height, the identifier of the block at that block height, and the number of chunks contained in the snapshot. The *snapshot body* consists of multiple chunks of UTXOs that make up the UTXO set to be encoded. We limit chunk sizes to 1 MB in correspondence to Bitcoin's old maximum block size (cf. Section 2.2.2), which has proven to be transferable sustainably over the Bitcoin network. The *snapshot identifier* is a cryptographic hash value created over the snapshot's header and chunks to uniquely represent the snapshot in a succinct manner. The snapshot identifier is derived in a two-step procedure to improve the validation of obtained snapshots for joining nodes: First, the identifiers for the header and each chunk are derived by individually hashing them using Bitcoin's HASH256 function (i.e., SHA256 applied twice). Then, the snapshot identifier is derived by concatenating the header identifier and all chunk identifiers in order, and applying HASH256 to the result. Using this simple procedure, joining nodes can request the snapshot header and every individual chunk separately based on their own identifiers, but they can also validate these identifiers' correctness by recalculating the snapshot identifier without having to obtain the snapshot first.

Locally Persisted Information

Shifting to a snapshot-based synchronization process using CoinPrune enables established full nodes to completely prune older data. Namely, the full nodes can delete all blocks prior to the snapshot's block height, as they are not required anymore for the initial synchronization process. However, the full nodes must still supply joining nodes with the complete headerchain, i.e., all block headers starting from the genesis block, and the not yet prunable full blocks making up the chaintail. As a result, CoinPrune nodes have to store additional metadata that would otherwise only be stored in Bitcoin's block index. This metadata engulfs each block's identifier, header, block height, cumulative amount of PoW, number of contained transactions, and timestamp. Persisting this data, a recent snapshot, and the chaintail is now sufficient to securely bootstrap joining nodes.

5.5.2 Coordination of Established Nodes

CoinPrune establishes trust in recent snapshots by having miners mutually reaffirm their correctness on the blockchain. Joining nodes then use these reaffirmations as their trust anchor when deciding whether to synchronize based on a particular snapshot. As shown in Figure 5.2, CoinPrune miners coordinate based on *pulse blocks* P_i , which are issued in constant intervals Δ_P , e.g., every 10 000 blocks. Pulse blocks trigger the creation of a new snapshot and its subsequent reaffirmation phase 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 snapshot during Δ_R (cf. Section 5.8.2). 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 (cf. Section 2.4.3.1).

With this process, all CoinPrune nodes can successfully coordinate their activities. On the one hand, triggering pruning operations using pulse blocks at deterministic positions on the blockchain enables all CoinPrune nodes to derive *and reaffirm* the same snapshot. On the other hand, the pulse duration allows for tuning the pruning process such that its overhead does not outweigh the benefits for the overall network.

5.5.3 Bootstrapping of New Nodes

In this section, we present how newly joining full nodes can be bootstrapped using the snapshot-based synchronization provided by CoinPrune. Namely, we detail how joining nodes obtain and validate their snapshot, the headerchain, and the chaintail, and we elaborate how the nodes process this information to synchronize.

In a first step, a joining node has to *obtain a recent-enough snapshot*. The node has two potential sources for securely obtaining snapshots; it can either request their snapshot from its neighbors using off-chain P2P requests, or from external third parties, such as snapshot mirror servers. Since the on-chain reaffirmations allow the joining node to assess which snapshots are considered valid by the other nodes via

their cryptographic identifiers, the node can acquire the snapshot either way. Downloading the snapshot from a third party serves as a convenience solution, whereas the joining node can use the P2P-based method if it does not trust any particular third party. We further detail our adaptations to realize the P2P-based snapshot acquisition in Bitcoin in Section 5.7.2. As the second step, the joining node *downloads and verifies the headerchain* from its neighbors to learn about the blockchain branch with the most PoW in it, as is already done using Bitcoin’s headers-first download [Bit15a]. In the third step, the node *initializes its UTXO set* based on the UTXOs contained in the snapshot instead of downloading and processing all full blocks. As the fourth and last step, the 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 short full synchronization phase, the joining node additionally inspects the coinbase fields of all blocks in the chaintail for reaffirmations of its applied snapshot. If the node learns that its snapshot was the most-reaffirmed one during the reaffirmation window Δ_R and that it was reaffirmed at least k times (cf. Section 5.5.2), it accepts the snapshot and is now synchronized. Otherwise, the node aborts and obtains a different snapshot from another source, e.g., by connecting to a new set of neighbors.

In conclusion, CoinPrune’s snapshot-based synchronization process allows bootstrapping nodes despite skipping all full blocks prior to the last valid snapshot.

5.6 Handling Application-Level Data

We now present extensions to CoinPrune that enable the dedicated handling of higher-level semantics of blockchain data. We first show how an additional *local obfuscation* of UTXOs, which is analogous to the obfuscation used by redaction juries in RedactChain (cf. Section 4.3), effectively removes illicit content also from the UTXO set without losing compatibility to Bitcoin (Section 5.6.1). Afterward, we present how CoinPrune can be extended to preserve benign application-level data stored in OP_RETURN payloads (Section 5.6.2).

5.6.1 Obfuscation of Illicit Data from the UTXO Set

We have already applied the concept of output obfuscation to counteract unwanted blockchain content insertion, which was conceptualized by Florian et al. [FHBS19], in Section 4.3.6.2 to help automate the redaction of address-manipulated transactions in RedactChain. However, manipulated on-chain addresses are not only present on the blockchain, but they are also included in the UTXO set and pollute it indefinitely. In this section, we thus extend upon the prior output obfuscation and adapt the idea to also protect the UTXO set maintained by CoinPrune nodes. After presenting the construction of our obfuscation scheme and discussing that it does not impede the utilization of the UTXO set to validate pending transactions, we outline and contextualize limitations of this approach before discussing future potentials.

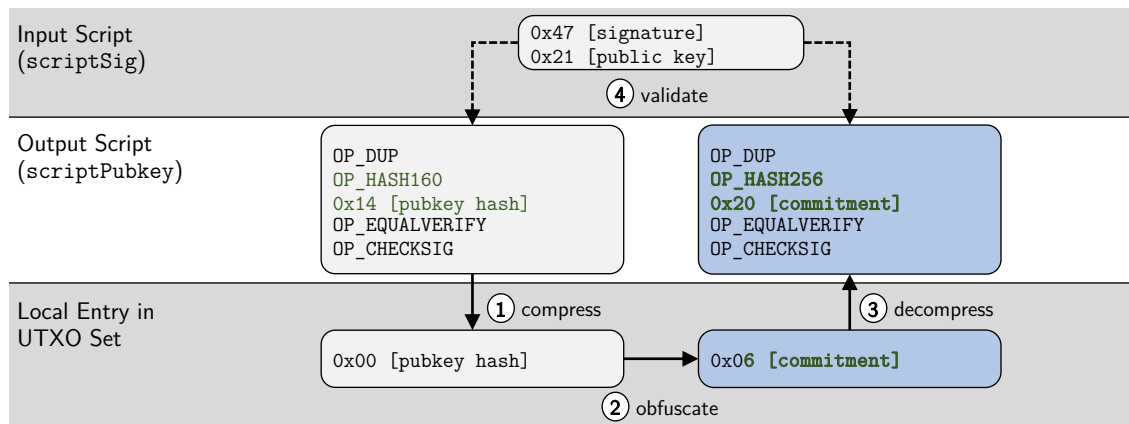


Figure 5.3 CoinPrune can be extended to obfuscate compressed UTXOs (here P2PKH) by applying HASH256 to the stored value and decompress obfuscated output scripts analogously to how RedactChain operates (cf. Section 4.3.6.2).

5.6.1.1 Construction

In Chapter 4, we already proposed the obfuscation of transaction outputs at two stages. On the one hand, hardened on-chain addresses (cf. Section 4.2.5) constitute a robust alternative to address-manipulable on-chain addresses. On the other hand, redaction juries can effectively remove unwanted content from the blockchain in an automated fashion by hashing the potentially-manipulated on-chain addresses during the redaction process (cf. Section 4.3.6.2).

We can extend CoinPrune with a corresponding *local obfuscation*, where CoinPrune nodes obfuscate all UTXOs they are aware of instead of altering on-chain information. Figure 5.3 illustrates the local obfuscation of UTXOs. Essentially, a CoinPrune node stores hash values of on-chain addresses in their UTXOs instead of storing the on-chain addresses directly. However, the obfuscation has to be adapted to the layout of UTXOs, which are *compressed* representations of transaction outputs (cf. Section 2.3.5). In the example shown in Figure 5.3, a node obfuscates a UTXO derived from a P2PKH transaction output, but any output script relying on hash-based on-chain addresses can be handled analogously. In more detail, CoinPrune’s obfuscated UTXO set management entails four steps:

First, the CoinPrune node *compresses* a transaction output to be added to the UTXO set (Step ①). As we described in Section 2.3.5, (legacy) Bitcoin nodes already compress UTXOs when possible and prepend the compressed value with one additional byte indicating the compression case applied (one of the values 0x00 to 0x05). Otherwise, the node keeps the uncompressed script and prepends it with its length plus six (to account for the six compression cases).

CoinPrune nodes additionally *obfuscate* hash-based on-chain addresses within compressed UTXOs (Step ②); to do so, they store the HASH256 value of the recorded on-chain address instead of the address itself. Consequently, the node cannot restore the value due to the pre-image resistance of SHA256. Besides P2PKH outputs, CoinPrune nodes can also obfuscate P2SH, P2WPKH, and P2WSH outputs (cf. Section 2.3.3.3) analogously. Combined, these script types account for 99.2%

of all UTXOs at the time of our measurements (cf. Section 5.9.2). To integrate CoinPrune’s obfuscation into Bitcoin, we introduce four new compression cases corresponding to which type of output script was obfuscated. For instance, a compressed-and-obfuscated P2PKH output is marked by setting the first byte to `0x06` instead of `0x00` (cf. Figure 5.3). Correspondingly, uncompressed UTXOs now have the script’s length plus ten (instead of six) as the first byte. This shift is possible as CoinPrune’s updated UTXO set management does not interfere with the purely local handling of the UTXO set by legacy nodes.

Before validating a pending transaction’s inputs, the node has to *decompress* the referenced compressed-and-obfuscated UTXOs first (Step ③). By reading the first byte, CoinPrune nodes know which type of transaction output was compressed and can restore it. Additionally, they can infer whether or not the UTXO was obfuscated; if so, the node can add the required `OP_HASH256` operation on the fly to validate against the obfuscated instead of the plain on-chain address in the next step.

Finally, the node *validates* the pending transaction’s input script against the now-decompressed UTXO (Step ④). By accounting for obfuscation in the previous step, the same input script satisfies both output-script templates. Hence, transaction creators can remain oblivious about whether full nodes apply obfuscation.

5.6.1.2 Limitations

CoinPrune’s obfuscation scheme relies on the fact that most UTXOs are compressible and hold a hash-based value instead of, e.g., raw public keys. In these cases, we can add our simple `HASH256`-based obfuscation without further changes, as a transaction input accessing such a UTXO already has to provide the pre-image matching the hash value recorded in the UTXO. However, albeit deprecated, Bitcoin still permits P2PK and P2MS outputs, which contain one or more raw public keys [Bit10d]. CoinPrune cannot support the obfuscation of these output patterns, as adding this support would require extending the corresponding input scripts, which would prevent CoinPrune from being retrofittable to Bitcoin as a velvet fork. Similarly, we do not support the obfuscation of non-standard transaction outputs, which cause pending transactions to be rejected by Bitcoin’s full nodes, but are accepted when a miner directly includes them in their blocks (cf. Section 2.4.2). Hence, our scheme also cannot support the obfuscation of non-standard transaction outputs. Still, as discussed in the last section, our obfuscation scheme covers 99.2% of the UTXO set.

5.6.1.3 Future Potentials

In the case of a 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 outputs can improve the coverage of our obfuscation scheme, analogously to the stricter checks we proposed to improve RedactChain’s applicability in Section 4.3.6.2. Second, preexisting UTXOs that currently cannot be obfuscated by CoinPrune could be rewritten to enable obfuscation. This rewriting is not

possible at the moment, as legacy transaction creators would not be aware that they would have to adapt their input scripts. Finally, if the full nodes reached consensus to allow carefully rewriting the UTXO set to a certain extent, future work could investigate further means to counteract UTXO set pollution. For instance, nodes could completely drop UTXOs flagged for encoding illicit content or repossess old UTXOs that only hold negligible values and thereby bloat the UTXO set. However, such modifications would have strong implications for Bitcoin and demand a comprehensible yet minimal rule set for allowed modifications that has been approved by a large majority of nodes.

5.6.2 Preservation of Application-Level Data

As we have discussed before, the blockchain-backed data management has an inherent duality (cf. Section 3.4): Besides the insertion of unwanted blockchain data with its associated negative consequences, short chunks of non-financial data, especially `OP_RETURN` payloads, can prove beneficial and enable higher-level applications. However, full nodes identify `OP_RETURN` transaction outputs as inherently unspendable and therefore do not include the corresponding payloads in their UTXO set (cf. Section 2.3.5). Hence, block-pruning schemes, such as CoinPrune, would remove this data, potentially breaking higher-level applications relying on `OP_RETURN` payloads.

Thus, a second CoinPrune extension provides an *application data storage* that realizes a lookup table of past `OP_RETURN` payloads together with their relevant context. This context entails the identifiers of the transaction holding the `OP_RETURN` output and the encapsulating block. As nodes still maintain the headerchain, this information suffices to associate application data with a specific transaction and its position on the blockchain. This way, CoinPrune retains information about, e.g., the ordering and inclusion times of `OP_RETURN` payloads of a given application.

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 5.5.1). CoinPrune nodes hence can derive a cryptographic identifier for a snapshot of the application data storage analogously to the snapshots' identifiers. To preserve on-chain storage, CoinPrune miners derive a combined identifier for both, the current snapshot and the corresponding state of the application data storage, and include that single identifier in their blocks. This combined identifier is obtained by concatenating the snapshot identifier and the application data storage's identifier, and then applying `HASH256` to the result.

Takeaway. Overall, these CoinPrune extensions allow for handling both desirable and unwanted non-financial data when pruning obsolete data. Hence, measures to counteract the growing impact of increasing data volumes are not necessarily in direct conflict with the applications for blockchain-backed data management.

5.7 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. We separately discuss required adaptations to what data is stored on the blockchain (Section 5.7.1) and backwards-compatible changes to Bitcoin’s P2P protocol (Section 5.7.2).

5.7.1 Additional On-Chain Data

CoinPrune miners publish snapshot reaffirmations on Bitcoin’s blockchain (cf. Section 5.5.2). At the same time, the miners must preserve compatibility with legacy full nodes in order to avoid breaking changes. In the following, we discuss how CoinPrune achieves its deployability via a velvet fork (cf. Section 5.1.1.3).

CoinPrune miners reaffirm a snapshot’s validity by including a *reaffirmation tag* in the coinbase field of their blocks. As we discussed in Section 3.3.1, the coinbase field can *optionally* hold up to 96 Byte of arbitrary data. The coinbase field’s contents are not subject to validation by the full node. Hence, legacy nodes ignore any reaffirmation tag. Furthermore, CoinPrune nodes never reject any block with a reaffirmation tag they find to be invalid. Instead, having multiple miners redundantly record the reaffirmation tag for the last pulse block’s snapshot (cf. Section 5.5.2) allows CoinPrune nodes to rely on positive-only feedback. Thereby, valid snapshots are expected to outpace invalid ones based on the number of reaffirmation tags recorded on the blockchain (cf. Section 5.8.2). As a result, our block-pruning scheme fulfills the requirements for a gradually deployable velvet fork (cf. Section 5.1.1.3). To prevent CoinPrune nodes from confusing reaffirmation tags with other coinbase payloads, reaffirmation tags should follow a recognizable pattern. Using a unique prefix and separators in the coinbase field, such as `CoinPrune/[snapshot_identifier]/`, helps keeping CoinPrune’s messages distinguishable from other applications’ coinbase payloads. This way, CoinPrune can potentially coexist with other applications using the coinbase field, such as merged mining [JZS⁺17].

5.7.2 Adaptions to the 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 snapshot. For this purpose, we extend the data dissemination within the Bitcoin network (cf. Section 2.5.3) while ensuring that CoinPrune nodes remain compatible with legacy nodes.

We extend Bitcoin’s current data dissemination capabilities with a new message type `getstate` that operates similarly to `getblocks` (cf. Section 2.5.3). During CoinPrune’s adapted initial synchronization process, joining nodes first obtain their snapshot before they download the headerchain and chaintail (cf. Section 5.5.3). To do so, a joining node first sends a `getstate` message to its neighbors to learn about

available recent snapshots. Each neighbor answers with an `inv` message that contains the hash values of the header and chunks of their most recent and successfully reaffirmed snapshot as `state` objects. The joining node can derive each advertised snapshot's identifier from these `inv` messages and determine which snapshot was advertised by most neighbors. Then, the node requests the chunks of this snapshot from its neighbors using `getdata` messages. For increased compatibility, we restrict chunk sizes to 1 MB, i.e., Bitcoin's old maximum block size (cf. Section 2.2.2). Finally, the node applies the snapshot once all chunks are available.

The `getstate` message is a new message type that is exclusive to CoinPrune and not understood by legacy nodes. To avoid sending new messages to legacy nodes, nodes signal CoinPrune support during the connection handshake using an additional service flag in the `version` message (cf. Section 2.5.2). If both nodes support CoinPrune, they can use `getstate` messages thereafter. Otherwise, CoinPrune nodes fall back to Bitcoin's legacy P2P protocol to ensure compatibility. During the initial synchronization process, CoinPrune nodes should focus on connecting to other CoinPrune nodes. If they only maintain a few connections to other CoinPrune nodes, they potentially limit their access to valid snapshots. Since a node's advertised services are also disseminated as part of periodically sent `addr` messages (cf. Section 2.5.2), the joining node can explicitly pick CoinPrune-supporting neighbors.

Takeaway. As CoinPrune only relies on the `coinbase` field and only requires minor changes to Bitcoin's P2P protocol that are in line with its built-in extensibility features, Bitcoin nodes can adopt CoinPrune without unwanted side effects.

5.8 Security Discussion

In this section, we assess the security properties of CoinPrune by discussing how it helps nodes bootstrap correctly (**G2**) based on verifiable snapshots (**G3**) that are maintained by an honest majority of CoinPrune miners. We first show that the on-chain reaffirmations reliably reference snapshots from arbitrary sources to be used for bootstrapping joining nodes more efficiently (Section 5.8.1). Second, we discuss the influence of the positive-only feedback created by CoinPrune miners mutually reaffirming the current snapshot has on its trustworthiness (Section 5.8.2). Finally, we discuss the P2P-level security of CoinPrune (Section 5.8.3).

5.8.1 Snapshot Validity

Joining CoinPrune nodes rely on the validity of on-chain reaffirmations to verify that they bootstrap using a correct snapshot. CoinPrune ensures that snapshots are correctly linked to the correct position on the blockchain via its pulses and the two-way binding between snapshots and the blockchain.

All CoinPrune nodes know which blocks are pulse blocks and which blocks may hold reaffirmation tags (cf. Section 5.5.2). This way, the nodes know to expect the availability of a snapshot corresponding to each pulse block unless they detect that the

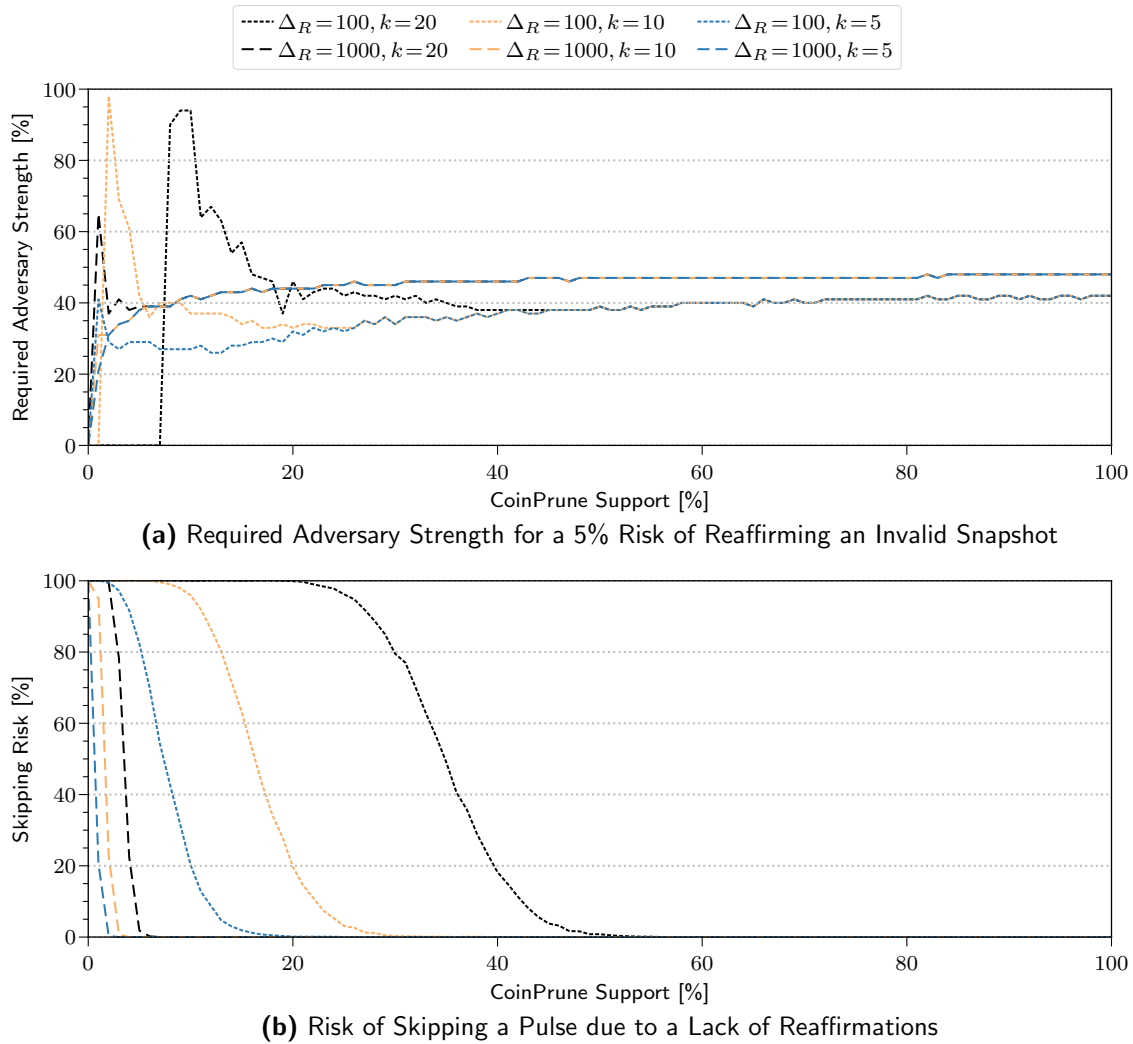


Figure 5.4 A strong support of CoinPrune improves its resilience against adversaries. Relying on larger reaffirmation windows further increases this resilience, independent of the minimum required number of reaffirmations for a snapshot to be accepted.

pulse was invalid, i.e., no snapshot reached enough reaffirmations. Furthermore, each snapshot references its corresponding block height in the snapshot header. Hence, nodes can reliably assess which part of the blockchain is covered by the snapshot.

To ensure that snapshots are referenced correctly on the blockchain, CoinPrune uses a cryptographic hash function to obtain snapshot identifiers. In its layered approach, the snapshot header and the chunks comprising its body are first hashed individually before the concatenation of this list of hash values is hashed again using HASH256. This approach enables joining nodes to notice any manipulation of the header (including the associated block height) or the individual chunks when validating a snapshot's integrity against the pulse's reaffirmation tags during their initial synchronization. Hence, a malicious adversary can neither modify the encoded UTXO set nor the snapshot's position on the blockchain without changing the snapshot identifier. Thus, adversaries cannot deceive joining nodes to accept invalid or inconsistent snapshots, e.g., to reinstate already spent UTXOs.

Takeaway. The pulse-based coordination and the two-way binding between the snapshots and the blockchain allow CoinPrune nodes to assess a snapshot’s integrity and validity regardless of its origin, i.e., the Bitcoin network or snapshot-mirroring web-sites. Hence, the nodes can safely apply any valid snapshot they find to be sufficiently reaffirmed during the corresponding pulse’s reaffirmation window.

5.8.2 Reliability of Reaffirmations

In the previous section, we established that reaffirmation tags allow CoinPrune nodes to reliably obtain successfully reaffirmed snapshots. However, this reliability would be undermined if an adversary succeeded in correctly reaffirming an invalid snapshot. In Section 5.5.2, we introduced the acceptance threshold k and the reaffirmation window Δ_R to prevent an adversary from launching this type of attack. To investigate the resilience of CoinPrune’s reaffirmation process, we simulated a random mining process of 1000 active miners with a gradually increasing share of CoinPrune miners and an increasing adversarial influence among the CoinPrune miners.

For our simulation, we consider that a share of $0\% \leq f_C \leq 100\%$ miners support CoinPrune, and we increase f_C in 1%-steps. For each sampling point for f_C , we also sample a growing fraction $0\% \leq f_A \leq 100\%$ of adversary-controlled CoinPrune miners; again, f_A grows in increments of 1%. The sampling point $f_C = 25\%$ and $f_A = 10\%$ thus implies that 250 out of the total 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. For each scenario, we 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.

Figure 5.4 shows the result of our simulation; we depict 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 (Figure 5.4a) and the worst-case risk over all f_A that no snapshot was reaffirmed successfully (Figure 5.4b).

Our results show that a longer reaffirmation window of $\Delta_R = 1000$, i.e., roughly one week of real-world time in Bitcoin, results in a good resilience against adversarial CoinPrune miners. Namely, the adversary has to control at least $f_A \geq 46\%$ of all CoinPrune miners for a moderate CoinPrune support of $f_C \geq 31\%$ to be able to comprise 5% of the snapshots on average. This threshold is further raised to $f_A = 48\%$ for full CoinPrune support, regardless of the acceptance threshold k .

For the smaller reaffirmation window, $\Delta_R = 100$, 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 disproportionately 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 accordingly. This observation implies that a *dynamic* approach, where Δ_R and Δ_P are adapted 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 a small pulse duration, e.g., $\Delta_P = \Delta_R = 100$. If f_C was sufficiently large during the last pulse, we can relax CoinPrune’s aggressiveness and set, e.g., $\Delta_R = 1000$ and $\Delta_P = 10\,000$. Similarly to the deployment of P2SH [And12a], CoinPrune can also conduct an *initial* voting phase to sense a sufficient support f_C before starting the first pulse.

Takeaway. Pulses can be configured to ensure a good resilience against malicious adversaries, especially with stronger CoinPrune support. An initial voting can ensure a sufficient level of support before starting the first pulse. Further, shorter and more frequent pulses can reduce an adversary’s influence during phases of lower support.

5.8.3 Peer-to-Peer Attacks

After having established that snapshots are resilient against manipulation and reaffirmations are robust against dishonest minorities, we now assess to which extent the adversary can target CoinPrune nodes via our adaptations to Bitcoin’s P2P protocol.

As we presented in Section 5.7.2, our changes to the P2P protocol are designed with backwards compatibility in mind (**G4**). By having nodes announce their support for CoinPrune during the handshake via a service flag in the `version` message, CoinPrune nodes can fall back to the legacy protocol when establishing connections to legacy Bitcoin nodes. As a result, any messages that are not understood by legacy Bitcoin nodes are only exchanged between CoinPrune nodes.

CoinPrune only introduced one such additional message type, and one new object type. First, joining nodes send an additional `getstate` message to learn about the snapshots advertised by their neighbors. Second, contacted nodes answer with an `inv` message containing the newly introduced `state` objects. Both are direct extensions of message and object types already existing in Bitcoin’s P2P protocol; thus, considerations regarding the resilience against DoS attacks or Eclipse attacks [HKZG15] also translate to CoinPrune’s extensions. Namely, assuming that a joining node is served the correct headerchain and chaintail, i.e., a subset of the required information for Bitcoin’s legacy synchronization process (cf. Section 2.5.4), it can reliably identify whether its neighbors advertise invalid snapshots and restart the synchronization process with a new set of neighbors if required. When we reconnected a commodity PC (cf. Section 5.9.1) 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 150 s on average (excluding six measurement points where the client did not connect to ten other nodes). Hence, resetting with new neighbors is a feasible means to react to identified issues on the P2P layer, even when accounting for early neighbor disconnects.

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.

Takeaway. CoinPrune’s adaptations to Bitcoin’s P2P protocol provide joining nodes with a secure in-band option to acquire a recent snapshot to synchronize from.

5.9 Performance Evaluation

We now demonstrate that CoinPrune enables massive performance savings for both joining and synchronized full nodes (**G1**). After describing the setup of our testbed (Section 5.9.1), we present the storage savings achieved for all nodes (Section 5.9.2). Further, we show that CoinPrune massively reduces the required traffic and synchronization times for joining nodes as well (Section 5.9.3).

5.9.1 Testbed Setup for Synchronization Measurements

To assess the performance of CoinPrune, we implemented a proof-of-concept prototype based on Bitcoin Core v0.17.1 and released it as open source [MKP⁺20b]. We run our 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 being created every 10 000 blocks up to a block height of 640 000 (Jul 20, 2020) with 1000 additional blocks (roughly one 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 checking the coinbase field for the snapshot identifier to be able to use Bitcoin’s real blockchain for our measurements.

5.9.2 Storage Savings

In the following, we evaluate CoinPrune’s impact on the nodes’ storage requirements. First, we present the massive storage savings achievable by transitioning from Bitcoin’s full-chain synchronization process to CoinPrune’s snapshot-based synchronization. Afterward, we show that CoinPrune’s extensions for advanced data management, i.e., obfuscating unwanted data from the UTXO set and the additional application data storage, can be deployed without impeding the achievable savings.

Savings from Pruning

All CoinPrune nodes can prune older blocks in exchange for maintaining a recent snapshot and auxiliary information regarding the headerchain. To analyze the potential for reducing storage requirements via CoinPrune, we depict the main contributors to Bitcoin’s storage demands over time compared to the serialized snapshot

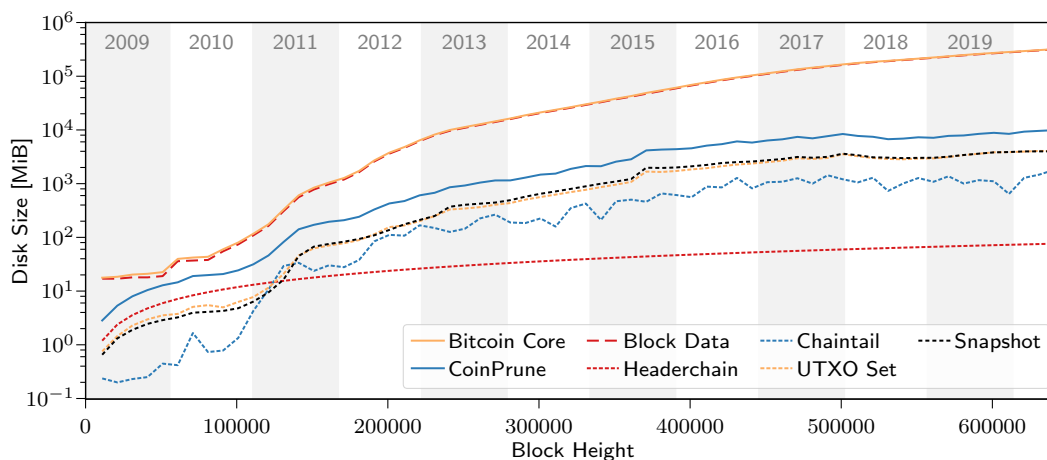


Figure 5.5 The storage requirements inflicted by CoinPrune’s adapted synchronization process are dominated by its snapshots’ sizes, which are tightly coupled to the size of Bitcoin’s UTXO set at the time of snapshot creation. Compared to a traditional synchronization using the full blockchain, CoinPrune nodes can reduce their storage requirements by two orders of magnitude when they only have to store a snapshot instead of all legacy full blocks.

and headerchain in Figure 5.5. We assess Bitcoin’s storage requirements based on its `blocks` and `chainstate` folders. The heavily dominating `blocks` folder contains the raw block data, information required to rewind blocks efficiently (e.g., in case of a resolved fork), and the block index. The `chainstate` folder contains 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 and auxiliary information to reconstruct the block index comes at only negligible costs of 125 Byte per block, resulting in a serialized 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 at higher block heights. In our latest measurement, CoinPrune enables a Bitcoin full node to save 302.53 GiB of storage. These savings account for a decrease of two orders of magnitude, with the potential for becoming even larger as the blockchain grows.

Overhead of Extensions

In Section 5.6, we presented two extensions to CoinPrune that allow considering higher-level data semantics. The first extension derives snapshots from obfuscated UTXOs to prevent the extraction of unwanted non-financial data encoded in UTXOs and the second extension is the application data storage, which persists `OP_RETURN` payloads that may be crucial for Bitcoin-backed applications but which would otherwise be pruned by CoinPrune. In the following, we analyze the feasibility of these extensions (**G5**) by assessing their additional storage requirements.

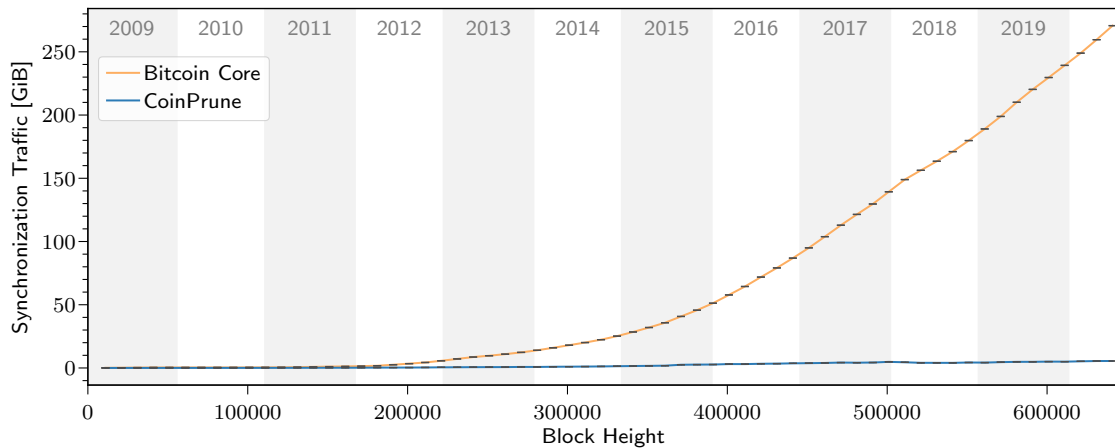


Figure 5.6 As joining nodes can obtain a snapshot and omit the majority of legacy blocks during the initial synchronization process, CoinPrune allows for bootstrapping new nodes with vastly reduced amounts of generated traffic, thereby unburdening all involved nodes.

Overhead of Content Obfuscation. The most recent snapshot in our measurements (corresponding to block height 640 000) contains roughly 66.2 million UTXOs. Our obfuscation scheme (cf. Section 5.6.1) can obfuscate 99.2 % of these UTXOs, the vast majority of which hold P2PKH or P2SH outputs. However, the obfuscation of these scripts increases the snapshot size, since obfuscated values have a length of 32 Byte instead of the 20 Byte required to store on-chain addresses in the clear. Due to this increase, 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, we serialized all `OP_RETURN` transaction outputs up to the block height of our most recent snapshot according to our scheme presented in Section 5.6.2. This serialized application data storage has a total size of 9.20 GiB and contains roughly 45.2 million entries with an average size of 219 Byte each. Hence, by pruning old blocks while preserving application data specifically, CoinPrune continues to support higher-level applications at feasible costs.

Takeaway. CoinPrune enables vast storage savings of over 85 % in our measurements. Our proposed extensions to handle non-financial data only slightly reduce these savings and are well worth the added protection and functionality, respectively.

5.9.3 Evaluation of Synchronization Performance

Pruning obsolete blockchain data does not only relieve established full nodes from storage depletion. Especially joining nodes benefit from the snapshot-based synchronization enabled via CoinPrune. In the following, we evaluate to which extent CoinPrune can reduce the required traffic and the initial synchronization time compared to Bitcoin’s current synchronization process.

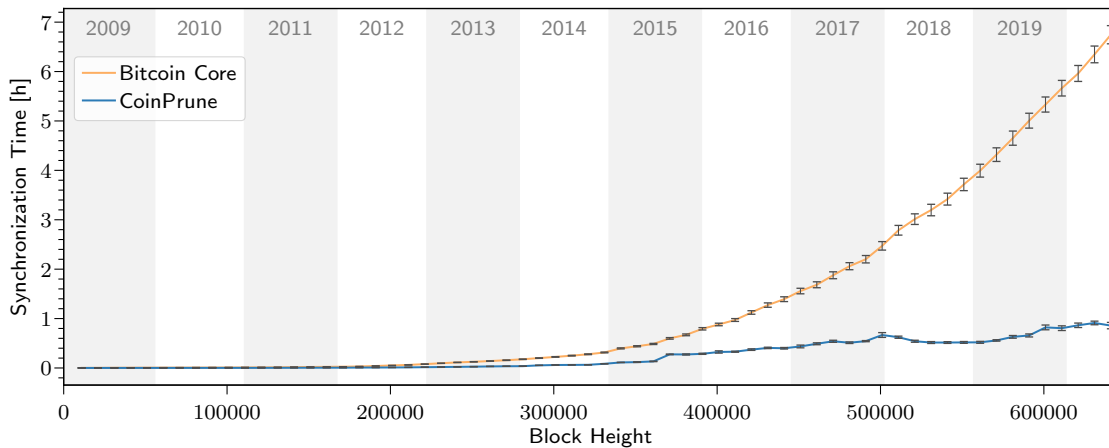


Figure 5.7 CoinPrune’s snapshot-based synchronization allows joining nodes to synchronize significantly faster than using Bitcoin’s full synchronization. However, the snapshot size continues to have a noticeable impact on the synchronization performance.

As shown in Figure 5.6, the reduced storage requirements directly translate to a reduction in traffic required to synchronize with the Bitcoin network. For instance, when 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 create 270.70 GiB of traffic during the bootstrapping process. Over the whole blockchain, achievable savings average at 92.99%. Overall, joining nodes have to obtain two orders of magnitude less data during the initial synchronization process using CoinPrune in our most recent measurements.

A similar trend can be observed for the overall synchronization time of joining nodes. Here, we measured the time it takes a joining node to obtain and verify the snapshot, the headerchain, and the chaintail. Figure 5.7 shows that CoinPrune improves synchronization times compared to Bitcoin’s full synchronization over the blockchain’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 5.2.2), joining nodes still must replay the whole transaction graph. CoinPrune’s snapshot-based synchronization enables joining nodes to skip this step as well. Consequently, synchronization times could be reduced to 51 min compared to 7 h when using standard Bitcoin in our latest measurements. This time increases when a node joins toward the end of a pulse due to a longer chaintail. However, Figure 5.7 indicates that the initial synchronization performance is vastly improved even in this case. This time saving is especially beneficial as the initial synchronization is considered one of the major scalability concerns [Lop20, CDE⁺16].

Takeaway. In summary, our results show that the snapshot-based synchronization of CoinPrune unburdens both established and joining full 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 and, hence, help mitigate the problem of growing data volumes in permissionless blockchains.

5.10 Conclusion and Future Work

This chapter focused on the prevalent problem of continually increasing data volumes in permissionless blockchain systems (**P2**, cf. Section 1.1.2). Especially established and long-running systems, such as Bitcoin, are subject to increasingly prohibitive blockchain sizes and corresponding negative impacts on storage and traffic requirements as well as prolonged synchronization times for newly joining nodes. While several improvements with the goal of countering this development have been deployed for Bitcoin, they only have low effectiveness. Consequently, new blockchain systems have been proposed with built-in pruning capabilities, but corresponding changes turn out not to be retrofittable to Bitcoin without creating permanent forks.

We proposed CoinPrune to remedy this situation. CoinPrune is designed as a Bitcoin-compatible block-pruning scheme that relies on periodically created snapshots. We tailored CoinPrune to be gradually deployable to Bitcoin via a velvet fork and thereby demonstrated that blockchains can successfully be retrofitted to address arising challenges (**Q2**, cf. Section 1.2). To this end, CoinPrune-supporting full nodes do not reject blocks containing invalid snapshot references, but the CoinPrune-supporting miners cooperate (**P3**) by mutually reaffirming correct snapshots and outperforming attempts of interference by dishonest minorities (**P1**). Our simulation shows that our scheme achieves security properties comparable to Bitcoin when 1/3 of nodes in the Bitcoin network support CoinPrune.

CoinPrune allows all nodes to completely prune obsolete blockchain data, which reduces the nodes' required storage capabilities from 312 GiB to 10 GiB (a decrease of 302 GiB) in our latest measurement, with potentially even larger saving potentials as the blockchain grows. Correspondingly, joining nodes can reduce their synchronization time by 77% in our measurements.

Finally, CoinPrune remains aware of higher-level data semantics via its extensions. On the one hand, the obfuscation of UTXOs encoded in a snapshot helps prevent the impact of unwanted blockchain content (**Q1**). On the other hand, the application data storage allows preserving intended blockchain data required for the proper operation of blockchain-backed applications (**Q3**, **P4**). Reducing a blockchain's storage footprint generally creates new capacities for deploying features that come with an overhead, e.g., the hardened on-chain addresses we presented in Section 4.2.5.

We envision future work to further improve CoinPrune, its deployment, and its handling of non-financial data. First, periodically deriving snapshots creates an additional overhead for CoinPrune nodes. Even though pulses can be prolonged to reduce the impact of this per-snapshot overhead, any optimization reducing the burden for full nodes increases the expected acceptance of CoinPrune. In our proof-of-concept implementation, for example, each snapshot is created from scratch, where an incremental snapshot management promises to simplify the snapshot generation. CoinPrune's resilience could be further improved as well. For instance, CoinPrune nodes may hold multiple recent snapshots available to improve their flexibility to react to errors. However, this measure implies additional overhead for CoinPrune nodes, due to maintaining more snapshots and a longer chaintail, and further analyses are required to assess how the nodes should manage their available snapshots

strategically. Finally, future work should incorporate higher-level application data more tightly in the design of block-pruning schemes. On the one hand, full nodes routinely pruning obsolete data creates the potential for making careful decisions to explicitly prune unwanted content as well, instead of only obfuscating it. Moreover, acceptable application data may become obsolete, and thus prunable, as well; however, additional coordination is required to define corresponding pruning rules. On the other hand, other applications may require fine-grained queryability also of older data and future work should investigate whether such applications can operate reliably when reducing data availability to the application data storage and a selection of full nodes voluntarily maintaining a full blockchain copy to respond to queries.

6

Blockchain-based Bootstrapping of Anonymity Services

For the final contribution of this dissertation, we demonstrate that a blockchain-backed data management holds opportunities for novel applications. Namely, we present a fully decentralized platform for bootstrapping anonymity services that benefits from the pre-established trust into long-running permissionless blockchains. Our bootstrapping platform, AnonBoot [MPBW20a], operates on top of Bitcoin and maintains a Sybil-resistant repository of privacy peers that are willing and eligible to contribute to operating different privacy services. From this peer repository, users can either locally draw random sets of privacy peers, e.g., to establish circuits for communication via an anonymity network such as Tor [DMS04], or they can request the establishment of ad-hoc privacy services that are based on mixing networks [Cha81, ZGH⁺15, ZMH⁺18, MM18] of independently drawn privacy peers.

In the following, we first motivate the need for bootstrapping anonymity services in a decentralized manner and why permissionless blockchains are a crucial building block for such an infrastructure (Section 6.1). Afterward, we derive design goals for the required bootstrapping platform (Section 6.2). We then continue with presenting the design of AnonBoot. After giving a design overview (Section 6.3), we detail how AnonBoot's peer repository is maintained on top of Bitcoin (Section 6.4) and how that repository is used for bootstrapping anonymity services (Section 6.5). Next, we discuss how different anonymity services can be realized using AnonBoot (Section 6.6). Subsequently, we evaluate our platform regarding its security (Section 6.7) and performance (Section 6.8). Finally, we discuss related work (Section 6.9) before concluding this chapter (Section 6.10).

6.1 Motivation

In this dissertation, we focused on identifying and alleviating current drawbacks stemming from using permissionless blockchains as a publicly writable, immutable, and ever-growing data storage. Given the requirements for moderating content being persisted on such a blockchain (cf. Chapters 3 and 4), dealing with growing sizes of massively replicated blockchain histories (cf. Chapter 5), and other drawbacks [BBSU12], one could ask: Which applications or use cases *do* warrant relying on blockchain-backed data management (**P4**, cf. Section 1.1.2; **Q3**, cf. Section 1.2)?

We have already shown in our prior contributions, namely RedactChain (cf. Section 4.3) and CoinPrune (cf. Chapter 5), that the characteristic properties of permissionless blockchains enable the coordination of independent parties. In this chapter, we transfer these insights and not only consider blockchain-intrinsic use cases, i.e., coordinating redactions and block-pruning activities. Instead, we investigate to which extent also external applications can benefit from this coordination channel.

For this purpose, we consider one class of applications that typically requires interaction between independent peers: *anonymity services*. Examples of anonymity services are anonymous communication networks based on onion routing, such as Tor [DMS04], message shuffling networks, such as Chaum mixnets [Cha81], or decentralized tumblers used to re-anonymize cryptocurrency users [ZGH⁺15, ZMH⁺18, MM18]. All these services heavily rely on coordinating multiple *anonymity peers* to increase the users' privacy (**P3**).

While various works have proposed and investigated secure building blocks for implementing anonymity services in the past, those works typically overlook the initial bootstrapping of the anonymity peers. Oftentimes, anonymity services are proposed under the assumption that its anonymity peers do not collude, e.g., due to their operators' presumed real-world reputation. However, this perceived reputation does not always warrant trust, as evidenced, e.g., by numerous alleged scams regarding cryptocurrencies [ZMH⁺18, Bad14, VM15] and the need for manually reporting [Tor14] or actively probing [CPPK11, WKM⁺14] bad peers in the Tor network (**P1**).

Hence, a bootstrapping process for anonymity services that retains their distributed nature is currently lacking. In the following, we thus investigate how permissionless blockchains can be utilized to close this gap, i.e., how a blockchain-based bootstrapping process can retain the desirable decentralization of current anonymity services. However, coordinating external applications via a permissionless blockchain also increases the number of transactions. Hence, we also have to assess how anonymity services can be bootstrapped with a low impact on the underlying blockchain (**P2**).

6.1.1 Problem Analysis

In this section, we revisit existing anonymity services (Section 6.1.1.1) and, from this analysis, we derive the need for creating a fully decentralized bootstrapping process for such services (Section 6.1.1.2).

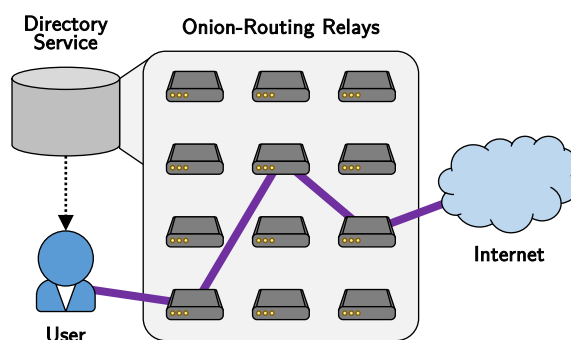


Figure 6.1 Anonymity networks, such as Tor, provide a repository of onion-routing relays for anonymous communication via the Internet. The user obtains a list of available relays from a directory service and locally chooses a random sample of relays to establish a circuit using layered encryption to communicate indirectly with the target Internet service.

6.1.1.1 Available Anonymity Services

Internet users frequently rely on distributed anonymity services to increase their privacy, and thereby outsource their privacy management. In the following, we identify and discuss three categories of such services: *anonymity networks* for browsing the Internet, *message shuffling networks*, and *cryptocurrency tumblers*.

Anonymity Networks. Privacy-aware Internet users refrain from accessing online services directly, as they may fear that an adversary monitors which servers they contact. *Anonymity networks* allow these users to remove this linkability and provide low-latency and anonymous Internet communication. Typical anonymity networks, such as Tor [DMS04], rely on *onion routing* to reach this goal. Figure 6.1 illustrates how Tor operates as an example of an onion routing-based anonymity network. The user first establishes a *circuit*, i.e., a cascade of intermediate peers, and then tunnels all traffic through this circuit. All traffic is sent under a *layered encryption* such that intermediate nodes of the circuit can gradually decrypt the traffic, i.e., each node can only lift one layer at a time. This way, any intermediate node can only infer their successor and predecessor node in the circuit. Establishing a circuit of at least three nodes then implies that no node can independently identify the sender nor receiver of any given connection: The circuit’s first node, the *guard node*, directly interfaces with the user but does not know their traffic. The last node, the *exit node*, only sees the user’s traffic in the plain but does not know to which user that traffic belongs. All other *middle nodes* neither learn the user’s identity nor their traffic.

The user creates their circuits locally at random, but they also consider performance metrics, such as the nodes’ available bandwidth [Tor06], as well as node-specific policies, e.g., exit nodes only performing requests to certain ports on the user’s behalf [DMS04]. Tor provides the information required to build circuits via a *directory* that is maintained by exceptionally trusted *directory servers* [DMS04]. These, at the time of writing nine [Tor09], directory servers are vetted by the Tor project maintainers, and users must trust that the directory servers do not collude [PRR09].

Message Shuffling Networks. Long before the recent proliferation of anonymity networks, David Chaum introduced *message shuffling networks* [Cha81], which can

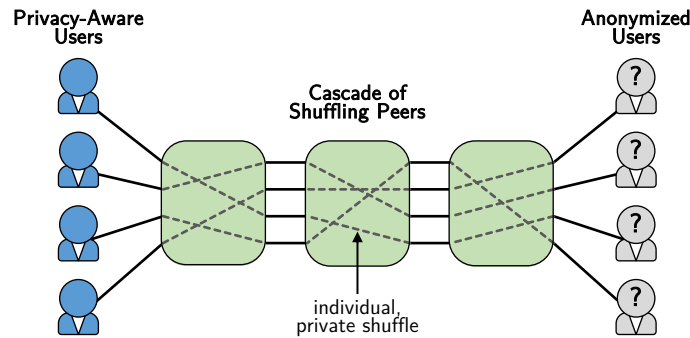


Figure 6.2 Message shuffling networks consist of a cascade of shufflers, who obliviously shuffle the encrypted input messages of multiple users in private before passing them to the next shuffler. Using a layered encryption prevents both shufflers and external observers from re-identifying which user submitted which message originally.

be used to realize sender-anonymous mail systems, e.g., to protect whistleblowers from retribution. Figure 6.2 shows the basic user interaction with a message shuffling network. Similarly to anonymity networks, users relay their messages through a cascade of known *shufflers*, which again involves a layered encryption. However, *multiple* users shuffle their messages through the *same* cascade of shufflers to achieve a vastly reduced overhead. These shufflers hence, one after another, receive the batch of encrypted messages, of which they can lift only the outermost encryption layer. After decrypting the message batch, each shuffler obliviously shuffles the messages and forwards the result to the subsequent shuffler. Therefore, shufflers are unable to correlate other shufflers' input and output batches. As long as one shuffler remains honest, no passive adversary can deanonymize the users from now on.

However, message shuffling networks are often prone to active attacks, such as denial of service (DoS) or replacing encrypted messages [DDM03]. Furthermore, adversaries can easily operate all shufflers of a message shuffling network at low costs, since those networks are fixed and small in size. Hence, users need to have a way to root trust in any message shuffling network they might submit their messages to.

Cryptocurrency Tumblers. As we have discussed in Section 2.1, traditional permissionless blockchains can only provide pseudonymity instead of true anonymity. To overcome this limitation and deter curious blockchain observers from tracking their financial activities, users started to make use of *cryptocurrency tumblers*, which we also refer to as *cryptotumblers*. Cryptotumblers pool the funds of multiple users seeking to increase their privacy and then pay out random coins of the same value to each user. As a result, blockchain observers remain oblivious as to which user now owns which coin. Cryptotumblers evolved over time, yielding different generations and flavors to appropriately address users' security and privacy concerns. Initial cryptotumblers were centralized and thus require users to trust that the service operator neither steals their funds nor discloses their shuffling history at any later point. Series of alleged scams [ZMH⁺18, Bad14, VM15] then emphasized the need for further technical protection, e.g., holding the cryptotumbler accountable [BNM⁺14]. The first generation of *decentralized* cryptotumblers enabled the users to jointly simulate a centralized tumbler by creating one large transaction where blockchain observers

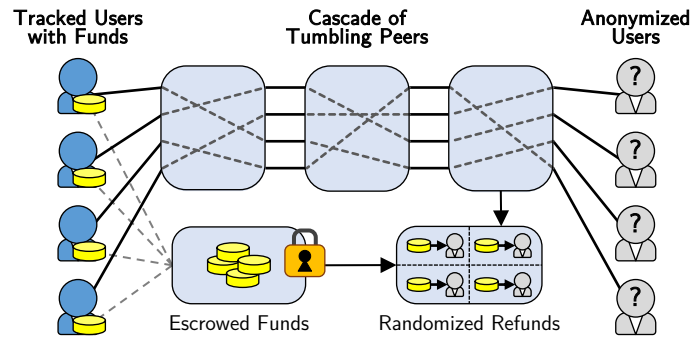


Figure 6.3 Distributed cryptocurrency tumblers have a similar goal to message shuffling networks, as they attempt to prevent blockchain observers from linking traceable coins to their owners. CoinParty [ZMH⁺18], as an example of a decentralized cryptocurrency tumbler, thus relies on an extended message shuffling network. Each user escrows a coin of the same value to a Bitcoin address that is jointly controlled by the tumbling peers using threshold cryptography. Instead of general messages, the tumbling peers then shuffle user-submitted fresh Bitcoin addresses and refund each user’s escrowed coin to a random other user.

cannot link an output to a specific user’s input anymore [Max13a, RMSK14]. As all users have to agree before they can mix their coins, this approach eliminates the risk of theft present in centralized cryptotumblers. However, single users can stall the mixing operation; the other users must be able to detect this case and then repeat the mixing without the misbehaving user [RMSK14].

Hence, another branch of cryptotumblers aims for providing a decentralized but dedicated mixing service [ZGH⁺15, ZMH⁺18, MM18], i.e., these cryptotumblers mix users’ funds on their behalves without reintroducing the risks of centralized services. One example of these cryptotumblers is CoinParty [ZGH⁺15, ZMH⁺18], which enables the decentralized mixing of bitcoins as illustrated in Figure 6.3: The tumbling peers first escrow the users’ coins using threshold cryptography (cf. Section 4.3.3.2), i.e., only a majority of tumbling peers can jointly decide how to spend the escrowed coins to mitigate theft. Similarly to Chaum’s shuffling networks, the tumbling peers then obviously shuffle user-submitted fresh Bitcoin addresses before transferring one coin per shuffled output address. CoinParty additionally protects this shuffling phase by introducing secret-shared checksums for user-submitted information, which allows holding misbehaving mixing peers accountable, e.g., when trying to replace a user’s submitted Bitcoin address with one under the control of a malicious tumbling peer. However, this additional protection can only tolerate adversaries controlling a share $f_S < 1/3$ [ZMH⁺18] of the tumbling peers due to the involvement of threshold cryptography (cf. Section 4.3.3.2).

6.1.1.2 The Need for Trustworthy Bootstrapping

Our discussion of current distributed anonymity services (cf. Section 6.1.1.1) reveals a common theme among those services: While the existing distributed anonymity services are secure, users are lacking means to assess the trustworthiness of the peers jointly providing the service. All services rely on the participation of a sufficient number of *independent*, i.e., honest and non-colluding, peers.

We can generalize distributed anonymity services for the scope of this chapter as follows. We assume a group of *privacy-aware users*, who seek to utilize an anonymity service that increases their privacy on their behalf. To provide sufficient security and privacy guarantees, the users require that multiple independent operators of *privacy peers* jointly offer the distributed anonymity services. As we have discussed in Section 6.1.1.1, the provision of these services can be based on one of two models: On the one hand, anonymity services such as Tor provide a pool of available privacy peers and the users *locally draw a random subset* of these peers to help them increase their privacy. The peers can be identified as untrustworthy by either active probing [CPPK11, WKM⁺14] or explicit user reports [Tor14]. On the other hand, services based on shuffling networks, e.g., cryptotumblers, consist of a *fixed set of connected* privacy peers that accept input from multiple users at once. Due to a limited scalability of network sizes of the latter category [ZMH⁺18], we assume that only a few privacy peers (e.g., <100) provide the services to potentially much larger user groups. These anonymity services are thus prone to Sybil attacks, where one adversary can control a large fraction of privacy peers with low effort.

Hence, privacy-aware users need a truly decentralized and secure means to ensure that distributed anonymity services are bootstrapped in a trustworthy manner. On the one hand, the users must be enabled to *securely discover* available privacy peers for anonymity services that rely on local selection by the user. On the other hand, they must be able to *establish trust* in the faithful setup of distributed anonymity peers consisting of a fixed set of privacy peers. In the latter case, multiple users with comparable privacy requirements must be able to discover or bootstrap anonymity services compatible to those requirements. A trustworthy bootstrapping procedure should furthermore *incentivize* maintaining an honest majority among the available privacy peers even though a share of privacy peers must be assumed to act maliciously nevertheless and aim to, e.g., deanonymize users, stall the service, or inflict other damages such as theft through cryptotumblers.

In conclusion, privacy-aware users need to be ensured that they only utilize distributed anonymity services that act faithfully, i.e., the majority of the respective peers are honest. However, especially the setup and discovery of such services currently constitute weak points that adversaries could exploit to infiltrate anonymity services. Thus, the bootstrapping of such services requires further research. In this chapter, we hence investigate to which extent the users' trust into long-running permissionless blockchain systems can be transferred to making the bootstrapping of distributed anonymity services more trustworthy as well.

6.1.2 Contributions

We propose to seize the trusted infrastructure of established permissionless blockchains (**Q3**, **P4**) to solve the bootstrapping problem for distributed anonymity services we identified in Section 6.1.1.2. We thus introduce *AnonBoot* as a blockchain-backed medium for maintaining a Sybil-resistant repository of independent privacy peers that facilitates the indexing and bootstrapping of distributed anonymity services in a secure, unbiased, and transparent fashion (**P1**). Privacy peers join the

repository by periodically publishing advertisements that contain a small Proof of Work (PoW) on the host blockchain within a limited time frame. On the one hand, the operators of privacy peers need to periodically invest hardware resources into refreshing their interest to stay in the repository, which mitigates the risk of Sybil attacks (**P1**). On the other hand, all participants can locally derive AnonBoot’s full state by just monitoring the host blockchain for related messages (**P3**).

Hence, AnonBoot creates a Sybil-resistant index of privacy peers from which users can then request new anonymity services to be bootstrapped. Users can choose privacy peers or established anonymity services from this index to cater to their individual privacy requirements. We exemplarily build AnonBoot on top of Bitcoin to demonstrate its low requirements regarding its host blockchain, i.e., AnonBoot does not require sophisticated blockchain features, such as smart contracts, to operate. We released a prototype implementation of AnonBoot as open source under the MIT license [MPBW20b]. Our evaluation shows that AnonBoot scales to repositories of, e.g., 1000 privacy peers and large user bases with only low storage impact on its host blockchain (**P2**) and low, tunable costs for its participants.

6.2 Design Goals

To remedy the missing trustworthiness when bootstrapping distributed anonymity services we identified in Section 6.1.1.2, we aim to create a decentralized bootstrapping medium based on a permissionless host blockchain. To realize our bootstrapping medium, we identify the following main design goals and desirable features.

(G1) Trustworthy Bootstrapping. Our medium must provide technical means to establish trust in available anonymity services, and it hence must be trustworthy itself. To this end, a decentralized design is reasonable to eliminate the need for users’ trust in any dedicated medium operator. Hence, we set out to investigate the applicability of permissionless blockchains to provide the basis for this bootstrapping medium. Furthermore, the medium must mitigate Sybil attacks to prevent its infiltration through adversaries. Finally, the medium must still remain in control over the setup of the offered anonymity services.

(G2) Secure and Lightweight Service Discovery. Our medium must only relay users to privacy peers and services that have been bootstrapped in a trustworthy manner. Previous approaches have proposed piggybacking the node discovery for peer-to-peer systems onto a well-established decentralized medium such as IRC [KWSW07]. In the context of our envisioned blockchain-backed coordination, service discovery must limit its impact on the host blockchain to facilitate the adoption of the bootstrapping process.

(G3) Broad Applicability. In Section 6.1.1.1, we discussed the variety of existing distributed anonymity services. Consequently, we must account for this variety and enable users to discover and utilize different services for different applications. Finally, users should be able to use anonymity services corresponding to their individual privacy preferences.

(G4) Scalability. Sufficiently large user bases are crucial to achieving high privacy levels via anonymity services. Our medium must thus effortlessly scale to large numbers of users and privacy peers.

(G5) Operator Incentives. Current honest anonymity services are typically offered on a voluntary basis [DMS04]. However, if the effort of signaling honesty through our medium to publicly offer anonymity services becomes burdensome for operators, the number of volunteers might decrease. Hence, our medium must also consider the option to compensate for operators' efforts in its design.

With these goals in mind, we present the design of our proposed bootstrapping medium, AnonBoot, in the next sections.

6.3 AnonBoot Overview

In the following, we first give an overview of *AnonBoot*, our proposed blockchain-backed medium for bootstrapping distributed anonymity services in a trustworthy manner, before presenting its design in more detail in the subsequent sections. First, we provide the intuition behind AnonBoot's design (Section 6.3.1). Afterward, we give an overview of the steps involved in bootstrapping an anonymity service using AnonBoot (Section 6.3.2).

6.3.1 Design Intuition

The main goal of AnonBoot is to provide a medium for *securely* bootstrapping distributed anonymity services that typically consist of only a few *privacy peers*. AnonBoot relies on a permissionless *host blockchain* to maintain a robust distributed *state* about available privacy peers and bootstrapping requests without storing any potentially privacy-compromising information about its users.

Figure 6.4 illustrates how the host blockchain facilitates the bootstrapping of distributed anonymity services. Privacy peers have to periodically *advertise* their interest to help provide an anonymity service on the host blockchain. By requiring interested privacy peers to solve a PoW puzzle within a short time frame as part of this advertisement, AnonBoot establishes a *Sybil-resistant peer repository*. This way, the peer repository only contains recently active privacy peers and adversaries need to repeatedly invest resources in *maintaining* their influence rather than being able to increase it over time. From this peer repository, new anonymity services can be *bootstrapped* in a trustworthy manner **(G1)**. Depending on the anonymity services, users can either locally draw random privacy peers from the peer repository or they can request the establishment of a new anonymity service for shared use, e.g., a cryptotumbler instance. In the latter case, AnonBoot dynamically *elects* privacy peers based on their valid advertisements as well as further randomness drawn from the host blockchain. Thereby, AnonBoot prevents adversaries from manipulating the peer election process to gain an advantage over honest peer operators.

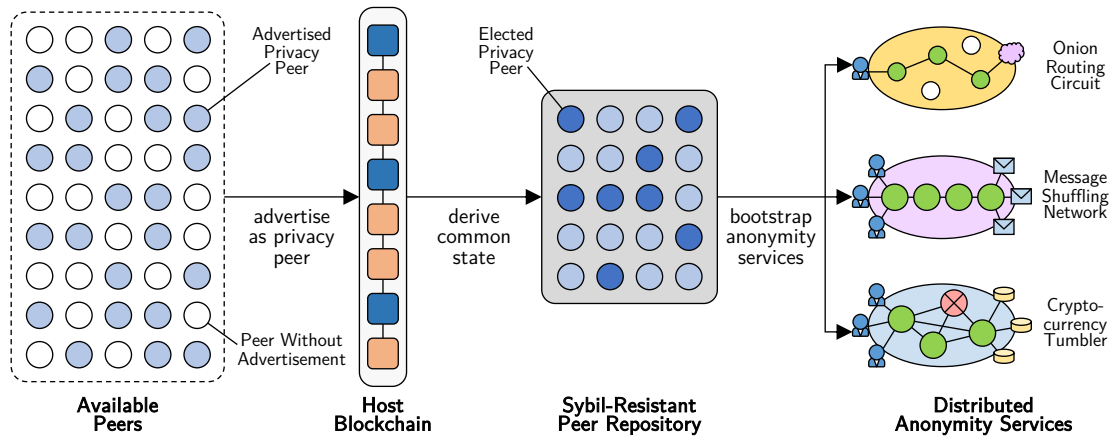


Figure 6.4 AnonBoot provides a medium for securely bootstrapping distributed anonymity services using a permissionless host blockchain. Interested privacy peers have to periodically refresh their membership in a Sybil-resistant peer repository by solving a PoW puzzle when advertising themselves on the host blockchain. From this peer repository, users can either directly draw peers locally, e.g., to establish an onion routing circuit, or request the bootstrapping of a shared anonymity service such as a message shuffling network or a cryptotumbler.

To interact with the host blockchain, all AnonBoot participants locally operate a *connector*. The connector publishes new messages to the host blockchain and monitors it for new events. Based on these events, the connector updates the local state of AnonBoot. In the remainder of this chapter, we detail how Bitcoin can be used as AnonBoot’s host blockchain despite its very restricted intended ways to insert application-level data to show that AnonBoot can operate on top of most blockchains. Furthermore, the Bitcoin network provides an especially strong trust anchor for operating AnonBoot due to its roughly 15 000 active nodes [Bit13c].

Next, we give an overview of the steps involved in using AnonBoot to bootstrap and discover distributed anonymity services.

6.3.2 Overview of Bootstrapping Steps

The bootstrapping process enabled by AnonBoot consists of four different steps. Figure 6.5 illustrates these steps and their interplay: First, in Step ①, privacy peers *advertise* themselves on the host blockchain. Subsequently, in Step ②, users *request* that new anonymity services are bootstrapped from a random subset of correctly advertised privacy peers. Next, in Step ③, all participants locally derive a common AnonBoot state. Finally, based on the derived state, users can either (Step ④_a) *locally select* privacy peers for personal anonymity services without the need for further synchronization with other participants, or (Step ④_b) privacy peers *bootstrap* a new shared anonymity service as previously requested. We now provide a more detailed overview of these individual steps.

Periodic Peer Advertisements. In Step ①, AnonBoot creates and maintains its peer repository by requiring privacy peers interested in providing anonymity services to periodically issue *advertisements* on the host blockchain. Peer operators need to

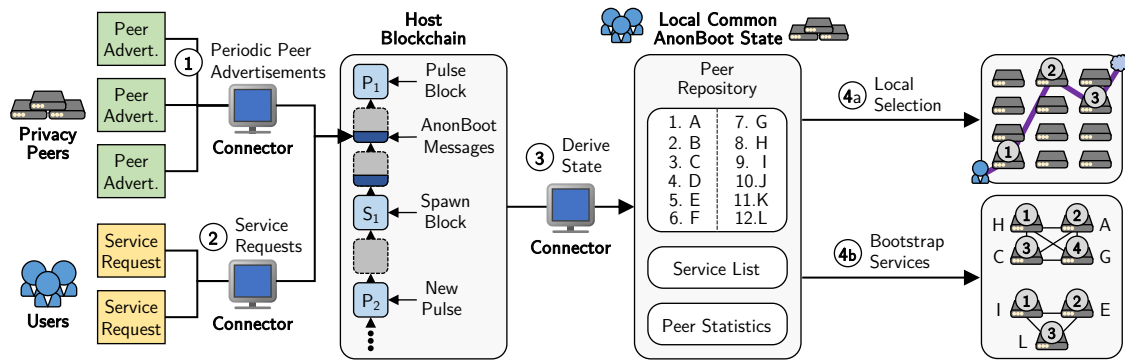


Figure 6.5 AnonBoot participants interact with the host blockchain using a connector. At the start of each pulse, privacy peers can ① advertise their willingness to contribute to anonymity services and users can ② request the bootstrapping of a shared anonymity service by storing corresponding messages ③ on the host blockchain. All participants can then ③ derive a common local state, which can be used for ④a local peer selection or ④b bootstrapping shared services.

periodically refresh their advertisements at the start of each refreshment period, or *pulse*, while solving a small PoW puzzle. This way, AnonBoot ensures that the peer repository remains Sybil-resistant, as peer operators need to invest their hardware resources at the start of each pulse to remain part of the peer repository. To mitigate any advantage adversaries may gain through dedicated mining hardware, the exact design of the PoWs puzzles is a crucial parameter of AnonBoot (cf. Section 6.7.1).

Service Requests. In Step ②, privacy-aware users may issue aggregatable on-chain *service requests* to request the bootstrapping of a shared anonymity service, e.g., a message shuffling network or a cryptotumbler, after a fixed-length negotiation phase. Service requests specify the type of the anonymity service as well as service-specific parameters such as minimum required sizes of anonymity sets.

State Derivation. In Step ③, all participants locally process the advertisements and service requests added to the host blockchain during the latest refreshment period to derive and verify AnonBoot’s current state. By locally processing all on-chain messages, the participants maintain a common *index of privacy peers* and a common *service list*. For processing requests, AnonBoot uses a randomized *peer election* process similar to RedactChain’s jury election (cf. Section 4.3.7.1) to select random subsets of compatible privacy peers, which then jointly provide the requested service. The peer election is based on a pseudo-random number generator (PRNG) that is seeded in a tamper-resistant way with randomness drawn from the host blockchain to enable all participants to locally derive the same service list. After discarding invalid or delayed messages, all participants obtain the same state, i.e., the peer index, service list, and statistics about previously discovered peers.

Local Selection and Service Bootstrapping. Step ④ finalizes AnonBoot’s bootstrapping process with two possible approaches for users based on the type of anonymity service they plan to use. Either, users perform an instant *local peer selection* directly based on the peer repository (Step ④a), e.g., to establish a Tor circuit. Alternatively, users browse the service list (Step ④b) for securely bootstrapped anonymity services (G2). Since privacy peers derive the same state as users, they

can check whether they were elected to provide a shared anonymity service and subsequently bootstrap these services by contacting other elected privacy peers. In both cases, communication is initiated through the AnonBoot connector, which then hands over the control entirely to the underlying anonymity protocol.

After this overview, we present the design of AnonBoot’s protocol in more detail in the following. First, we detail how participants coordinate via the host blockchain to maintain the peer repository and file service requests (Section 6.4). Afterward, we further present how AnonBoot makes use of the established common state to facilitate the bootstrapping of distributed anonymity services (Section 6.5).

6.4 Sybil-Resistant Index of Peers and Services

AnonBoot relies on its host blockchain to maintain its Sybil-resistant peer repository and to instantiate new anonymity services based on users’ requests. We now detail how AnonBoot can use Bitcoin as its host blockchain only using `OP_RETURN` transaction outputs, i.e., only making use of intended means to store its messages on the blockchain (cf. Section 3.3.1). All concepts carry over to other blockchains, especially to systems that can process arbitrary on-chain messages through smart contracts, e.g., Ethereum. First, we present the structure of AnonBoot’s Bitcoin-compatible messages (Section 6.4.1). Afterward, we elaborate on how AnonBoot enforces that participants have to periodically refresh messages to keep the peer repository Sybil-resistant and reduce the impact on the host blockchain (Section 6.4.2).

6.4.1 Message Types

AnonBoot participants solely coordinate via short messages that are persisted on the host blockchain and interpreted by the participants’ connectors to update their local state. We first give an overview of the general structure of AnonBoot messages (Section 6.4.1.1) before detailing both required message types, peer advertisements (Section 6.4.1.2) and service requests (Section 6.4.1.3).

6.4.1.1 Basic Message Layout

AnonBoot connectors coordinate via short messages that are persistently stored on the host blockchain. Even for restricted blockchain systems such as Bitcoin, all messages can be included using only the intended means of augmenting a transaction with a `OP_RETURN` output with a payload of up to 80 Byte (cf. Section 3.3.1). Figure 6.6 illustrates the basic layout of AnonBoot messages. All messages share a common header consisting of a protocol identifier, an identifier of AnonBoot’s protocol version, an indicator of the message type, and four reserved bits for potential future use. The protocol identifier is a fixed two-byte string (`AB`) to help distinguish AnonBoot messages from messages created by other services, as is common for `OP_RETURN`-based protocols [BP17]. The message type is used to distinguish *peer*

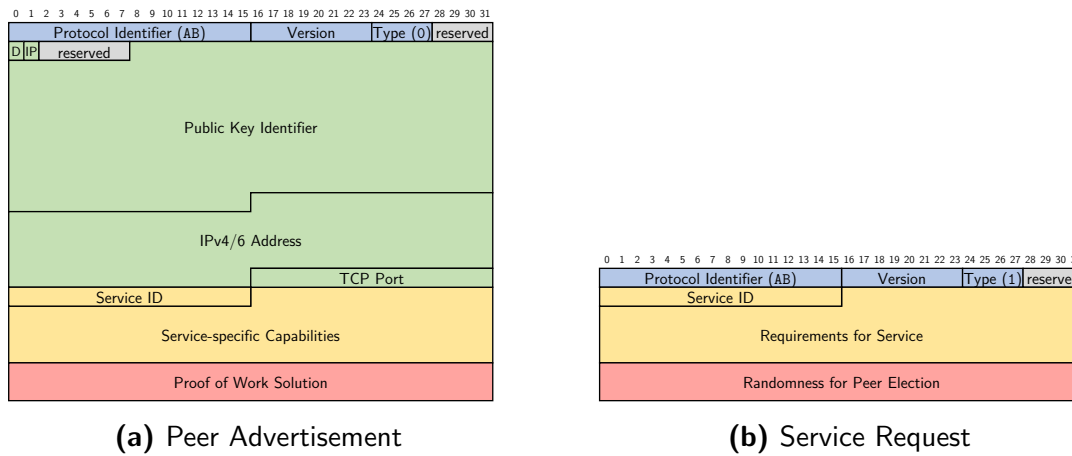


Figure 6.6 AnonBoot messages are stored on Bitcoin’s blockchain as OP_RETURN payloads. Peer advertisements convey peers’ contact information, capabilities, and the required PoW. Service requests trigger the bootstrapping of a service compatible with the user’s requirements and include a nonce for further randomness for the peer election process.

advertisements (Figure 6.6a) and *service requests* (Figure 6.6b), whose payloads are similar but not identical in structure. In the following, we discuss the similarities and differences between both message types in more detail.

6.4.1.2 Peer Advertisements

Privacy peers remain in the peer repository by periodically refreshing and publishing *peer advertisements* on the host blockchain. The payload of a peer advertisement contains three main components (cf. Figure 6.6a): (a) the peer’s *contact information*, (b) its *capabilities*, and (c) a *solution* of the PoW puzzle required for the advertisement to be considered valid. While sharing their capabilities and contact information allows coordinating the peer election (cf. Section 6.5.3), ensuring Sybil resistance via the PoW puzzle is crucial for AnonBoot’s promised security properties.

Contact Information. Publishing a privacy peer’s *contact information* enables other participants to contact it securely during the bootstrapping and service utilization. To this end, the privacy peer announces a cryptographic hash value of its connector’s public key as well as the IP address and port used to accept connections (cf. Figure 6.6a). The indirection through the connector enables a unified connection interface for all anonymity services supported by AnonBoot. However, if the advertised service’s contact information fits into the peer advertisement, the privacy peer may set the D-flag to indicate the direct reachability of the service, i.e., the connector can be bypassed. By setting the IP-flag, the privacy peer toggles whether it is reachable via IPv4 or IPv6, respectively. Similarly, we reserved six additional bits for future use to remain flexible regarding other formats of contact information.

Capabilities. The second component of a peer advertisement indicates the privacy peer’s *capabilities*. These capabilities entail a service identifier, which specifies the

anonymity service supported by the advertising privacy peer, and a set of service-specific capabilities. This design supports the integration of a diverse landscape of anonymity services (cf. Section 6.1.1.1) as well as future services and thus satisfies our goal of broad applicability (**G3**). These service-specific capabilities help users request services or locally select privacy peers that suit their individual needs. While smart contract-based host blockchains can process arbitrary messages and thereby enable the fine-grained expression of privacy peers' capabilities, the space limitations of Bitcoin's `OP_RETURN` payloads restrict this expressiveness. For instance, creating Tor circuits relies on potentially complex relay descriptors [Tor07] that easily exceed the available space and would impose a large overhead on the host blockchain otherwise. We make AnonBoot operable even in such restricted environments by allowing privacy peers to advertise coarse-grained capabilities as a browsing aid that is subsequently verified and refined via the participants' connectors.

PoW Puzzle Solution. Finally, privacy peers need to include the solution to a small *PoW puzzle* in their peer advertisements to thwart Sybil attacks. To be effective, the PoW puzzle must be cryptographically tied to the peer's identity as well as a recent point in the host blockchain to prevent an adversary from pre-computing or reusing peer advertisements. This way, the PoW puzzle is suitable to deter peer operators from creating a disproportional number of peer advertisements compared to their available hardware resources. We further discuss the impact of the chosen PoW scheme in Section 6.7.1.

6.4.1.3 Service Requests

Users issue *service requests* (cf. Figure 6.6b) to express that they want AnonBoot to bootstrap a new anonymity service corresponding to their requirements. Service requests closely resemble peer advertisements in their structure, but they do not contain contact information. Further, the remaining fields are interpreted slightly differently. On the one hand, the capability fields are repurposed to state the type of service to be bootstrapped as well as minimum requirements. AnonBoot allows users to only request distinct classes of services to prevent a highly fragmented service list. On the other hand, users do not solve a PoW puzzle in their service requests. Instead, users choose a random nonce, which adds further randomness for the peer election process when bootstrapping the requested service (cf. Section 6.5.3). This way, users can further thwart adversarial attempts to interfere with the peer election. A single service request will cause AnonBoot to instantiate the requested service to be used by an arbitrary number of users. Hence, AnonBoot easily scales to large user bases (**G4**). However, users questioning the existing requests' randomness can issue redundant service requests and thereby contribute to that randomness. AnonBoot aggregates redundant requests or similar requests superseded by service requests with stronger requirements, i.e., all requests' nonces influence the peer election, but only one service with the most restrictive capabilities of all aggregated service requests will be bootstrapped. At this point, we leave defining strategies to simultaneously instantiate multiple similar services as future work.

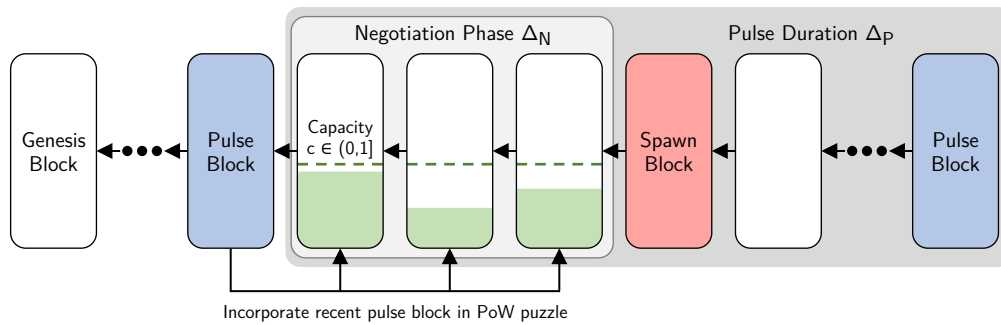


Figure 6.7 Every Δ_P -th block on the host blockchain is considered a pulse block and triggers privacy peers and users to issue their peer advertisements and service requests during the following negotiation phase of length Δ_N , which is concluded by a subsequent spawn block. Messages outside of the negotiation phase are invalid, and miners are advised to optionally not exceed a configurable per-block capacity for AnonBoot messages. The pulse block and spawn block provide further randomness for solving the PoW puzzles and for the peer election.

6.4.2 Pulse-based Message Release

Similarly to CoinPrune (cf. Chapter 5), AnonBoot relies on releasing its relevant messages in *pulses* after which messages become irrelevant. This way, AnonBoot implements a soft-state approach where privacy peers have to refresh their peer advertisements regularly. This approach ensures that the derived state remains fresh, and it provides a time frame during which the privacy peers have to solve their PoW puzzle (**G1**); furthermore, the pulses can be used to regulate the frequency at which new AnonBoot messages need to be stored on the host blockchain (**G2**).

Figure 6.7 illustrates AnonBoot’s pulse-based approach. A *pulse* has a *length* Δ_P , which is expressed in the number of blocks successfully added to the host blockchain between two pulses. The start of a pulse, indicated by the *pulse block* P , signals to all AnonBoot participants that all peer advertisements must be refreshed and that new service requests are now accepted. Once a new pulse block was mined, the privacy peers can start to solve the PoW puzzle for their peer advertisements based on the pulse block. Namely, the privacy peers solve the PoW puzzle while incorporating (a) their connector’s public key, (b) a reference to the pulse block to ensure the freshness of advertisements, and (c) a nonce that then solves the PoW puzzle. To extract the maximum randomness from the pulse block, AnonBoot can apply extraction techniques, e.g., as proposed by Bonneau et al. [BCG15].

For ideal fairness, all privacy peers would have the same time window for solving their PoW puzzle. However, AnonBoot must be able to cope with a potential backlog of valid peer advertisements because the miners of the host blockchain may prioritize other transactions over those containing AnonBoot messages. As a result, we cannot assume that all privacy peers have their peer advertisements included on the host blockchain immediately after solving their PoW puzzle. Thus, we tolerate peer advertisements to be delayed throughout a *negotiation phase* of length Δ_N after each pulse. This length should be chosen as short as possible to prevent a devaluation of the PoW provided by honest privacy peers, but it simultaneously should allow for including all anticipated peer advertisements in time even if single miners deliber-

ately ignore AnonBoot messages. Furthermore, the negotiation phase provides some tolerance against accidental blockchain forks (cf. Section 2.4.3). While peers must recompute their PoW if the host blockchain discards the pulse block, a fork does not require AnonBoot to skip an entire pulse. We refer to the first block following the end of the negotiation phase as the *spawn block* S . As we discuss further in Section 6.5.3, the spawn block provides further randomness for the peer election.

The tunable pulse duration with its associated negotiation phase also allows adjusting the burden put on the participants as well as the host blockchain in a fine-grained manner, and thus allows keeping the service discovery lightweight (**G2**). First, AnonBoot disincentivizes creating messages excessively, as honest peers will ignore all messages outside of a pulse’s negotiation phase. Second, increasing Δ_P without changing Δ_N reduces the number of messages required to maintain the peer repository, i.e., the costs are reduced for all privacy peers, without weakening AnonBoot’s Sybil resistance and only at the cost of the peer repository becoming less flexible. However, all AnonBoot messages are released in bursts at the start of each pulse. If these occasional message bursts prove to be burdening the host blockchain, AnonBoot-aware miners can follow an optional guideline to accept messages only up to a per-block *capacity* $c \in (0, 1]$ without impacting AnonBoot negatively. Furthermore, more awareness from miners on the host blockchain has the potential to further reduce costs of AnonBoot peers and thereby lower the bar for altruistic peer operators. Either through updated consensus rules or novel, AnonBoot-tailored blockchain designs, miners can be incentivized to reserve up to $c \cdot 100\%$ of their blocks during each negotiation phase for including AnonBoot messages at no costs. For instance, full nodes may then reject blocks that ignore a current backlog of pending AnonBoot messages. We further quantify how the host blockchain can steer the impact of AnonBoot in Section 6.8.2. While this approach requires that miners are not entirely oblivious of AnonBoot, it ensures that AnonBoot can operate at minimal costs without overburdening the host blockchain.

After presenting how the peer repository is maintained using PoW-based advertisements on the host blockchain, we now describe how AnonBoot bootstraps anonymity services from this repository.

6.5 Bootstrapping Secure Anonymity Services

All privacy peers that regularly refresh their peer advertisements are eligible for providing anonymity services via AnonBoot. In this section, we describe how AnonBoot facilitates bootstrapping anonymity services based on the current pulse and its resulting peer repository. After briefly describing how control is handed over from AnonBoot to its bootstrapped services (Section 6.5.1), we first consider how users can locally pick privacy peers directly from the peer repository (Section 6.5.2) before providing details on how AnonBoot elects privacy peers to bootstrap shared anonymity services (Section 6.5.3).

6.5.1 Connecting Users and Privacy Peers

AnonBoot provides only a medium for establishing and discovering trustworthy distributed anonymity services. Hence, AnonBoot must also enable users to contact the privacy peers that provide the requested services. In most cases, peer advertisements will announce how to reach the involved privacy peers' connector. In this case, both the user and the contacted privacy peers will have to perform a handover of control from the connector to the software implementing the anonymity service's protocol. During this handover, the user verifies the correctness of each privacy peer's contact information as published in the peer advertisement, especially whether they possess the right private key. In cases where indirection through the connector is undesired, privacy peers may use the `D`-flag (cf. Section 6.4.1.2) to signal that the contact information directly corresponds to the endpoint of its offered service. However, a Bitcoin-backed AnonBoot only supports direct advertisements if an `OP_RETURN` payload can hold all required contact information.

Depending on the particular anonymity service (cf. Section 6.1.1.1), users either contact (a) only one privacy peer, (b) all privacy peers of one anonymity service, or (c) may only indirectly contact subsequent privacy peers for security reasons, e.g., when establishing Tor circuits. In cases where a direct connection to all privacy peers is prohibited, the connection can be established gradually by incrementally contacting the next privacy peer's connector and performing the handover only after successfully establishing the connection to its predecessor. For instance, Tor builds circuits hop by hop [DM19], and thus users can contact the connectors of subsequent Tor nodes via partially established circuits, which aligns well with Tor's design [DM19].

6.5.2 Local Selection of Privacy Peers

The peer repository's Sybil resistance makes it a suitable replacement for centrally maintained directories, such as Tor's directory service (cf. Section 6.1.1.1). All AnonBoot users individually monitor the host blockchain for new peer advertisements, which enables them to instantly select privacy peers based on their local view of the peer repository. This way, local peer selection is independent of the pulse length. Furthermore, users may base their decisions on individual security and privacy preferences, e.g., they only select privacy peers with sufficiently recent peer advertisements, or they may locally keep track of peer statistics, such as the privacy peers' first occurrence or how regularly they refresh their advertisements.

When selecting privacy peers, the user verifies the correctness of those peers' advertisements and contacts their connectors. To this end, users only have to passively monitor the host blockchain for valid peer advertisements from the current pulse. Each privacy peer that (a) satisfies the user's privacy preferences, (b) performed a valid and fresh PoW, (c) is reachable via its connector's contact information, and (d) advertised a valid public key is eligible to be selected by the user. Ultimately, the user selects a random sample of eligible privacy peers and replaces any inaccessible privacy peers until the service can be provided correctly.

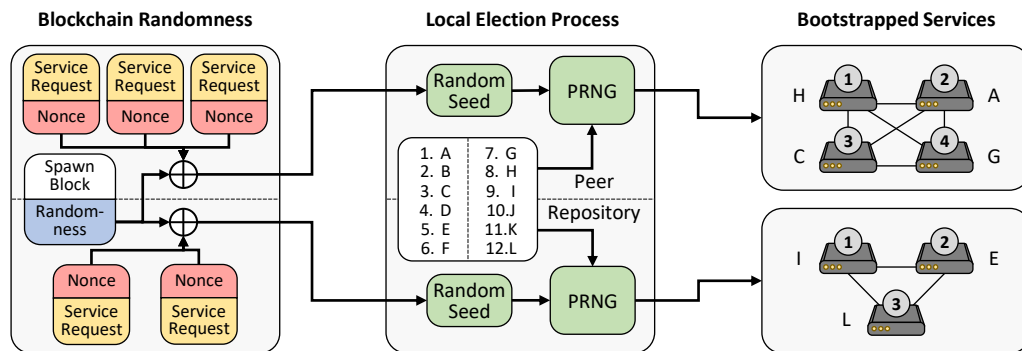


Figure 6.8 During the peer election, AnonBoot participants locally seed a PRNGs using the nonces from compatible service requests and randomness drawn from the pulse’s spawn block. This way, all participants locally compute the same service lists from the peer repository.

6.5.3 Service Requests for Peer Election

Besides enabling users to locally draw privacy peers directly from the peer repository, AnonBoot also facilitates the bootstrapping of shared anonymity services that are used simultaneously by multiple users. AnonBoot derives the demand for shared anonymity services from users’ service requests during the negotiation phase (cf. Section 6.4.1.3). Based on these service requests, we must ensure that peers are chosen randomly in a transparent manner to provide a secure bootstrapping process. Similarly to the jury election of RedactChain (cf. Section 4.3.7.1), we satisfy this requirement via a locally replicable *peer election* process that relies on a pseudo-random number generator (PRNG) and seeds derived from random values on the host blockchain. This way, all participants obtain the same list of elected peers for each distinct service request for subsequent coordination.

Figure 6.8 shows AnonBoot’s peer election process in detail. First, participants derive the common seed for the PRNG using two sources of randomness. The first source of randomness is the 8 Byte-long nonces users include in their service requests. We aggregate the nonces of all requests for the same service class during one pulse. With this approach, anonymity services can efficiently be bootstrapped from a single service request while simultaneously offering especially privacy-aware users the chance to directly influence the peer election’s randomness without spawning concurrent services that are potentially underutilized. As our second source of randomness, we consider the spawn block of a pulse, i.e., the first block *after* the pulse’s negotiation phase has concluded. This way, an adversary cannot craft nonces to bias the peer election without mining the spawn block, and we ensure that the randomness used for the PRNG seed incorporates fresh information from the host blockchain. All participants use this seed to initialize their local PRNG to perform the peer election for each aggregated service request, i.e., draw a pseudo-random sample of privacy peers from the peer repository that is compatible with the service request. A common ordering of the peer advertisements ensures that all participants select the same samples. The peer election allows all participants to compute the same service list and thus coordinate in a decentralized manner. As a result, AnonBoot helps users find the required entry points for using anonymity services,

and the privacy peers learn whom to connect to when being elected to join a specific anonymity service.

6.6 Realization of Use Cases

In this section, we present how the established distributed anonymity services we discussed in Section 6.1.1.1 can operate on top of AnonBoot; for this purpose, we assess the achievable benefits of relying on AnonBoot, the technical integration of the respective services, and how to optionally create financial incentivizes for honest privacy peers to participate (**G5**). First, we discuss how AnonBoot’s peer repository can establish a distributed directory service for anonymity networks (Section 6.6.1). Afterward, we discuss the bootstrapping of shuffling networks and cryptotumblers (Section 6.6.2), as both behave similarly in AnonBoot. Finally, we additionally outline how AnonBoot could also be used in the context of RedactChain (cf. Section 4.3) to enable normal users, and not only miners, to join redaction juries (Section 6.6.3).

6.6.1 Decentralized Onion Routing via AnonBoot

AnonBoot’s Sybil-resistant peer repository constitutes a cryptographically controlled replacement for otherwise logically centralized directory services as used, e.g., in distributed anonymity networks such as Tor. Hence, our approach is beneficial if users expect service operators to be corruptible or malicious. Nevertheless, users must still be able to make informed choices when establishing circuits. Furthermore, AnonBoot must account for the infeasibility and insecurity of users contacting all privacy peers of a circuit directly.

Benefits. Current anonymity networks rely on essentially centralized directory services. For example, Tor is pre-shipped with a hard-coded list of currently nine *directory authorities* [Tor09], which jointly maintain its service directory [Tor07]. This approach leaves current anonymity networks vulnerable to viable attacks on the directory service [DMS04]. Contrarily, AnonBoot allows creating a fully decentralized directory that is implicitly maintained through the host blockchain and locally verifiable by all AnonBoot participants. Based on this directory, users can locally select privacy peers for their circuits, as they currently do through Tor’s directory service.

Peer Advertisements. Privacy peers can use AnonBoot to advertise themselves as onion routers instead of being listed in Tor’s directory service. However, Tor’s directory service maintains *server descriptors* for each available onion router, which carry extensive meta information [Tor07] that, in most cases, cannot be encoded in a single `OP_RETURN` payload when AnonBoot operates on top of Bitcoin. Among this meta information is the peer’s contact information, cryptographic identity, available bandwidth, supported features, and exit policies, i.e., access control lists for connections to hosts on the public Internet [Tor07]. We thus make use of the peer advertisements’ capabilities (cf. Section 6.4.1.2) to encode an *overview* of the peers’

full meta information. This overview is a coarse summary of a privacy peer's *advertised* capabilities and should be indicative of its actual capabilities. Users can then browse available privacy peers based on these advertised capabilities without additional delays. When establishing a new circuit, the user should then request the chosen privacy peers' full server descriptors, verify that this descriptor matches the previously advertised capabilities, and check that the full descriptor is also compatible with the user's requirements.

Bootstrapping Phase. The circuits users establish within anonymity networks are intended to provide sender-receiver anonymity. Hence, a critical constraint is that users only communicate directly with the first peer of a circuit. AnonBoot naturally integrates with the resulting incremental circuit establishment of Tor [DM19]: The user incrementally establishes the next hop of their new circuit based on their selected peers' advertisements. They contact the new peer through the partially established circuit and attempt to hand over control to the Tor client through the connector. If the handover fails, e.g., due to an invalid advertisement, the user terminates the connection to that peer and selects a replacement privacy peer. Although an honest majority among privacy peers reduces the overhead of such security back-offs, enabling privacy peers to build up a positive reputation across consecutive peer advertisements promises to further reduce respective risks for users.

Incentives. If honest providers of onion routers must be compensated for investing their resources to repeatedly advertise themselves in AnonBoot, cryptocurrency-based service fees are a promising means for creating operator incentives. However, on-chain payments bear a high risk of implicitly recording information about users' circuits on the blockchain used for paying the fees. We thus propose that users and privacy peers create anonymous unidirectional micropayment channels [GM17] for this purpose. Although micropayment channels require an on-chain setup, users can protect their privacy due to the concurrent setup transactions of all users. This way, users can pay peers who advertise themselves via AnonBoot for their service.

In conclusion, AnonBoot provides a fully decentralized alternative to directory services for anonymity networks, which currently are provided by few, fixed authorities.

6.6.2 Shuffling Networks and Cryptocurrency Tumblers

AnonBoot's main advantage is to provide a medium for bootstrapping distributed anonymity services and to ensure their privacy peers' independence. Privacy-aware users can therefore achieve secure on-demand anonymization, e.g., for message shuffling or increasing their financial privacy.

Benefits. Distributed systems that outsource responsibility to a set of peers typically rely on secure multiparty computation (SMC) [ZMH⁺18, AMVA17] (cf. Section 4.3.3.2). Unfortunately, scalability limitations of those SMC protocols hinder distributing responsibility among large sets of privacy peers. Without carefully selecting the responsible privacy peers, insider adversaries thus can gain power and cause harm relatively easily. Aggravatingly, our considered use cases of anonymous

message disclosure and tumbling cryptocurrencies lack a trustworthy peer selection process, and adversaries are highly incentivized to attack such systems. For example, an adversary could easily spawn numerous interconnected privacy peers, and thereby mimic a distributed cryptotumbler, tricking users into relying on this scam. AnonBoot provides the ingredients to *cryptographically ensure* that an adversary cannot bootstrap malicious services due to its Sybil-resistant peer repository and locally verifiable peer election. Hence, privacy-aware users reduce their individual risks when utilizing distributed anonymity services bootstrapped via AnonBoot.

Peer Advertisements. The peer advertisements for shared anonymity services must enable the peer election process (cf. Section 6.5.3) to identify privacy peers that can jointly provide a requested service. However, we must expect that different types of shared anonymity services require advertising highly diverse capabilities due to the envisioned broad applicability (**G3**) of AnonBoot. Hence, AnonBoot does not consider the service-specific capabilities during the peer election but requires that the service identifier (cf. Section 6.4.1) is chosen such that it sufficiently categorizes privacy peers with compatible capabilities. Similarly to our previous use case, privacy peers should rather use the capabilities field in their peer advertisements to facilitate the users' browsability of anonymity services by advertising supported policies or security parameters.

Bootstrapping Phase. Privacy peers are partitioned by the service identifier in their peer advertisements to ensure that only compatible privacy peers bootstrap an anonymity service. By locally replaying the peer election process, each privacy peer gets to know (a) whether it was elected to provide a service, (b) which peers are elected to bootstrap the same service instance, and (c) the peer's logical position within the new service's network. This way, privacy peers can configure and bootstrap the anonymity service after the handover from AnonBoot. We conservatively declare services stale after a couple of pulses to mitigate the impact of churn and malicious services bootstrapped by chance. However, conceptually, AnonBoot also supports bootstrapping long-lived anonymity services.

Incentives. Since these use cases do not prohibit a direct connection between users and elected privacy peers, we can simplify our payment scheme proposed in Section 6.6.1 and instead require users to pay an upfront fee (e.g., as proposed by CoinParty [ZMH⁺18]). We argue that the increased security provided by AnonBoot is worth compensating the privacy peer's efforts of solving PoW puzzles.

In summary, AnonBoot addresses the users' skepticism that seemingly decentralized anonymity services may be controlled by a set of colluding attackers by mediating the bootstrapping of such services based on randomly electing independent privacy peers from its Sybil-resistant peer repository.

6.6.3 User-Centered Redaction Juries

AnonBoot's focus lies on bootstrapping anonymity services, but also other distributed services can profit from a decentralized bootstrapping medium. As an

example, we give an overview of how AnonBoot can be used to expand the pool of potential jury members in RedactChain’s redaction juries from recently successful miners (cf. Section 4.3) to also include other interested participants.

Benefits. RedactChain considers all recently successful miners eligible for being elected into a redaction jury (cf. Section 4.3.7.1). If RedactChain users come to the conclusion that the available pool of miners is insufficient, e.g., due to an unacceptable degree of effective miner centralization [MCR⁺20], AnonBoot can be integrated to allow other users to advertise their interest in participating in a redaction jury. This extension could hence increase the acceptance of RedactChain.

Peer Advertisements. RedactChain’s redaction juries are structurally very similar to distributed anonymity services; for instance, both RedactChain and CoinParty [ZMH⁺18] rely on threshold cryptography to provide their services. Consequently, the layout of peer advertisements for potentially joining redaction juries is similar to the layout we discussed in Section 6.6.2. In fact, these peer advertisements have a simplified layout, as redaction juries do not need to exchange specific capabilities.

Bootstrapping Phase. The bootstrapping of RedactChain’s redaction juries is analogous to the bootstrapping of shared anonymity services (cf. Section 6.6.2). Further, fully integrating an AnonBoot-based jury election process into RedactChain makes the indirection via a dedicated connector obsolete.

Incentives. In contrast to providing anonymity services for other parties, we view contributing to redaction juries as primarily motivated by the users’ best interest. Namely, RedactChain users have an interest to ensure that illicit content is redacted reliably so that they can operate full nodes without having to fear negative consequences (cf. Section 3.4.2). As such, we do not focus on financially incentivizing the users to become jury members but envision a sufficient intrinsic motivation.

Hence, AnonBoot’s applicability further extends beyond the scope of distributed anonymity services and may facilitate the bootstrapping of other distributed services, such as those based on threshold cryptography, as well.

Takeaway. AnonBoot provides a viable medium for bootstrapping anonymity services from a diverse set of available applications, as it simultaneously mitigates malicious influences and compensates honest operators of privacy peers. Furthermore, AnonBoot shows potential to improve the bootstrapping process in other scenarios as well, as exemplified by its applicability to RedactChain.

6.7 Security Discussion

We now assess AnonBoot’s robustness against adversaries. First, we discuss the implications of requiring privacy peers to solve PoW puzzles as part of their peer advertisements (Section 6.7.1). Afterward, we argue that active adversaries cannot effectively bias AnonBoot’s peer election process (Section 6.7.2).

6.7.1 Proof of Work Against Sybil Attacks

Requiring to show PoW in each peer advertisement hampers an adversary's effort to control large portions of the peer repository. However, the choice of the PoW scheme is paramount for AnonBoot's resilience against Sybil attacks. We thus highlight the need for an appropriate PoW scheme but leave its final instantiation to be adapted to users' needs in future work.

In particular, AnonBoot's PoW scheme must ensure that operators of privacy peers can only create peer advertisements at rates corresponding to the number of physical devices they control while not excluding honest peer operators who use commodity hardware. While specialized hardware is known to provide huge advantages for CPU-bound PoW schemes such as Bitcoin's HASH256-based scheme, memory-bound PoW schemes, such as the Ethash scheme Ethereum used before switching from Proof of Work to Proof of Stake [Woo14, Eth22], Cuckoo Cycle [Tro15], Equihash [BK17], or Manjaro's RandomX [Tev18, She19], are promising candidates to be adapted for utilization with AnonBoot. For instance, based on `openssl speed`, we observe that a server (2× Intel Xeon Silver 4116 CPU, 202 GB RAM) outperforms a commodity desktop PC (Intel Core 2 Q9400 CPU, 8 GB RAM) by two orders of magnitude for Bitcoin's PoW scheme. Further, Bitcoin mining hardware [Bit13b] reportedly outperforms our commodity PC by eight orders of magnitude, which clearly underlines the potential advantage of adversaries relying on specialized hardware to forge advertisements using CPU-bound PoW schemes.

Contrarily, measurements of Ethash (via `geth`'s CPU-based mining) and RandomX indicated that the same server only achieves a mere 7.5× (12.7×) speed-up over the desktop PC in terms of achievable hash rate using the respective PoW schemes. Thus, relying on memory-hard PoW schemes is preferable to prevent adversaries controlling powerful devices or, e.g., a botnet, from increasing their influence on the peer repository in an incommensurate manner [BK17].

Finally, we address the challenge of steering the PoW puzzles' difficulty to account for improvements in hardware capabilities. In contrast to cryptocurrency mining, AnonBoot's peer advertisements have no inherent concurrency, i.e., the size of the peer repository does not influence the required difficulty for the PoW puzzle. Assuming an honest majority, we can expect that privacy peers have an interest in keeping an appropriate PoW difficulty for security reasons. Thus, we can dedicate unused bits in the peer advertisements (cf. Section 6.4.1) to enable voting on increasing the difficulty. Privacy peers would then update their local threshold for accepting the PoW in peer advertisements based on votes of the (honest) majority.

Takeaway. Utilizing a simple CPU-bound PoW scheme for our puzzles would significantly impact AnonBoot's security. Contrarily, memory-bound PoW schemes constitute a more suitable building block to maintain a Sybil-resistant peer repository. As for existing systems, such as Tor or Bitcoin, the reliability of AnonBoot's peer repository then depends on maintaining an honest majority, either on a voluntary basis or through operator incentives. Finally, we can further leverage this honest majority to implement a self-regulated adaption of the puzzles' difficulty.

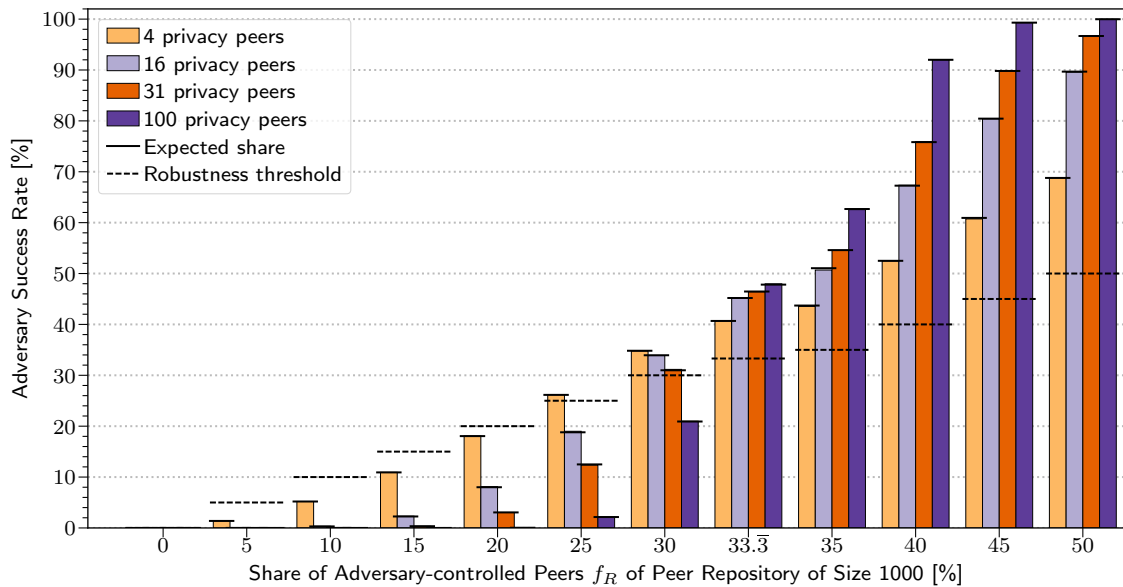


Figure 6.9 The success rate of an adversary to 1/3-infiltrate bootstrapped services by chance can be kept below the corresponding robustness threshold $T_{1/3}$ for $f_R \leq 30\%$, i.e., the peer election process remains robust for relevant scenarios involving SMC-based anonymity services.

6.7.2 Security of Bootstrapped Services

The core design goal of AnonBoot is to securely bootstrap distributed anonymity services. We have already argued in Section 6.7.1 that AnonBoot can maintain a Sybil-resistant peer repository, i.e., adversaries cannot control a disproportional fraction of the peer repository. However, adversaries can still enter the peer repository as long as they can create a valid PoW. We now highlight that AnonBoot’s local peer selection (Section 6.7.2.1) and its peer election process for shared anonymity services (Section 6.7.2.2) are robust against adversarial bias. Furthermore, we discuss that bootstrapped anonymity services can tolerate a share of adversarial privacy peers despite the additional indirection via the connector (Section 6.7.2.3) and potential attempts to launch denial-of-service attacks (Section 6.7.2.4).

6.7.2.1 Local Peer Selection

Anonymity services that rely on local peer selection only require the Sybil resistance provided by AnonBoot’s peer repository to operate securely (cf. Section 6.7.1). However, not all privacy peers are considered equal depending on the anonymity service: Tor users, for instance, only infrequently change their guard nodes compared to the other onion routers [LKBM17] and they are bound to only using exit nodes that support their requests [DMS04]. By encoding the privacy peers’ capabilities accordingly (cf. Section 6.4.1.2), users can account for these properties when establishing their circuits. The peer repository hence constitutes a secure alternative to current directory services provided by trusted third parties.

6.7.2.2 Robustness of Peer Election

In addition to ensuring the Sybil resistance of its peer repository for local peer selection, AnonBoot must ensure that shared anonymity services are bootstrapped in a secure manner. Hence, we now assess an adversary’s chances to gain control over an anonymity service via AnonBoot’s PRNG-based peer election (cf. Section 6.5.3).

We say that an adversary successfully *infiltrates* a shared anonymity service if they control a share $f_S \geq t_I$ of that service’s privacy peers, where t_I is an *infiltration threshold* that indicates at which point the adversary can defy the service’s underlying security guarantees. For instance, a malicious adversary infiltrates any anonymity service based on secure multiparty computation (SMC) once they control $f_S \geq 1/3$ of the service-providing peers [BOGW88]. Under this notation, we consider the peer election to be robust if adversaries cannot increase their chance of infiltrating bootstrapped services beyond their share f_R of privacy peers in the peer repository. More formally, assuming that no adversary’s share of controlled peers in the peer repository exceeds a threshold t_R , we define a *robustness measure* $\mathcal{R}(t_I, t_R) = 1 - \Pr(f_S \geq t_I \mid f_R \leq t_R)$. We further define that the peer election process is *robust* if and only if $\Pr(f_S \geq t_I \mid f_R \leq t_R) \leq t_R =: T_I$ holds, i.e., $T_I = t_R$ can be interpreted as a *robustness threshold* against t_I -infiltration. For instance, an adversary controlling up to $f_R = 10\%$ of the peer repository should only have a probability of $T_I = 10\%$ to t_I -infiltrate a shared anonymity service by chance.

Figure 6.9 highlights AnonBoot’s robustness regarding SMC-based anonymity services with $t_I = 1/3$ [BOGW88]. For n total privacy peers, these services can tolerate up to $\lfloor n/3 \rfloor$ adversary-controlled privacy peers. For requested services consisting of 4, 16, 31, and 100 privacy peers (i.e., $\lfloor n/3 \rfloor = 1, 5, 10, 33$ tolerable adversary-controlled peers), respectively, we measured the success of an adversary controlling a growing share f_R of the peer repository to infiltrate anonymity services by chance. To extract randomness from the pulse’s spawn blocks, we rely on the blocks’ Merkle tree roots. More secure randomness extraction can be achieved by applying more sophisticated dedicated *randomness extractors* [BCG15]. For our evaluation, we (a) assume a peer repository consisting of 1000 peers, (b) randomly elect peers for 100 000 anonymity services for each scenario, and (c) count the number of 1/3-infiltrated services. We also highlight the robustness threshold for comparison and provide the expected shares of 1/3-infiltrated services based on combinatoric considerations.

Our evaluation reveals two major findings: First, our peer election process is *fair* in that it almost perfectly yields the expected distribution of 1/3-infiltrated services when sampling from the honest and dishonest candidate peers uniformly at random. Second, except for tiny anonymity services, the peer election remains robust as long as the adversary controls $f_R \leq 25\%$ of the peer repository. For a growing adversary strength, AnonBoot cannot guarantee robustness, although larger anonymity services yield better protection as long as $f_R \leq 30\%$. For all $f_R \geq 1/3$, AnonBoot is not robust anymore, as the adversary can infiltrate most SMC-based services. However, in those cases, his control of the peer repository exceeds the infiltration threshold for SMC-based services; in this case, we already consider the peer repository insecure, i.e., the insecurity does not stem from the peer election process.

AnonBoot relies on randomness drawn from the host blockchain to seed its PRNG during peer election. Adversaries are thus tempted to influence the seed by interfering with the on-chain data with the goal of increasing their chances of infiltrating anonymity services. Our rationale for AnonBoot’s robustness only holds if we can effectively prevent such interference. As presented in Section 6.5.3, the seed derivation also covers user-submitted data to ensure that seeds are not entirely determined by the miners of the host blockchain. Since the seed derivation also covers the pulse’s spawn block, we further limit the adversary’s interference capabilities. Namely, the adversary must (a) successfully mine the pulse’s spawn block while (b) crafting this block to yield, in conjunction with the user-supplied randomness, a biased pre-image of a favorable seed that is (c) derived from a cryptographic hash function. Assuming that no adversary possesses the computing power to control the host blockchain, we deem this kind of attack economically infeasible, as honest mining is more profitable for the adversary. In the future, we could also adapt AnonBoot to consider multiple consecutive spawn blocks to further thwart the influence of adversaries.

6.7.2.3 Handover Process

For most anonymity services, AnonBoot requires indirection through the connector when first establishing connections. During this handover process, each participant’s connector has to authenticate all privacy peers based on the public key previously announced in the respective peer advertisements. Hence, users only connect to privacy peers controlled by operators that created valid and distinct peer advertisement. The adversary thus cannot launch Sybil attacks through this indirection.

6.7.2.4 Potential for Denial of Service

Due to our secure processes for local peer selection, robust peer election for shared anonymity services, and secure handover of control, the security of utilizing bootstrapped anonymity services only depends on the security guarantees offered by those services. While AnonBoot mitigates the infiltration of anonymity services, they are still prone to denial-of-service (DoS) attacks that would effectively prevent proper anonymization. However, we argue that the anonymity services currently covered by AnonBoot can cope with such attacks: First, AnonBoot allows for the efficient creation of circuits for anonymity networks via local peer selection. Hence, the limited influence of single stalling relays does not significantly impede the users’ privacy, as they can switch to another circuit in this case. Second, CoinParty, our investigated cryptotumbler, detects and excludes stalling peers as long as adversaries did not infiltrate at least $1/3$ of the peers of the CoinParty instance [ZMH⁺18]. Finally, while traditional shuffling networks do not provide protection against DoS attacks, extending them with the measures taken by CoinParty achieves the same level of protection. Thus, our peer election does not directly thwart DoS attacks, but their impact on our considered anonymity services is highly limited.

Takeaway. AnonBoot provides secure primitives for local peer selection and the unbiased election of privacy peers as long as the majority of the peer repository

contributes honestly to providing these services. Hence, AnonBoot is suitable to bootstrap trustworthy distributed anonymity services.

6.8 Performance Evaluation

We now evaluate the performance overheads created by using AnonBoot for bootstrapping distributed anonymity services to demonstrate its feasibility. Namely, we contextualize the additional data management and bootstrapping delay introduced due to the required synchronization with Bitcoin as the host blockchain (Section 6.8.1) before evaluating the additional storage requirements for full nodes of the host blockchain stemming from AnonBoot’s messages (Section 6.8.2).

6.8.1 Synchronization with Host Blockchain

To continually monitor AnonBoot’s state, participants should maintain a local copy of its host blockchain. However, participants only have to verify that pulse and spawn blocks as well as alleged AnonBoot messages are indeed part of the host blockchain. As pulses and their contained AnonBoot messages become obsolete over time, participants are only required to keep track of information similar to CoinPrune nodes (cf. Chapter 5), i.e., the headerchain and the full blocks of the relevant pulses. After verifying the relevant full blocks, the participants can further prune the full blocks except for the `OP_RETURN` payloads relevant to AnonBoot. The required number of blocks to process depends on the pulse length Δ_P , the validity period of single peer advertisements, and the maximum lifespan of bootstrapped anonymity networks. Even if bootstrapped services remain active indefinitely (cf. Section 6.6.2), new participants can still start synchronizing from a recent pulse before continuing to discover older services from older blocks in the background. After this initial synchronization process, the participants actively monitor the host blockchain for new AnonBoot messages. This overhead is negligible for Bitcoin, as new blocks are only mined every ten minutes (cf. Section 2.4.1).

This potentially slow block creation interval, however, introduces unavoidable delays for the bootstrapping of new anonymity services, as services are only created once a pulse’s spawn block has been added to the host blockchain (cf. Figure 6.7). For instance, a pulse length of $L_p = 12$ blocks with a negotiation phase of $L_N = 3$ blocks implies that privacy peers have at most 30 min to solve their PoW puzzle, but users have to wait up to 2 h in the worst case until their requested anonymity service starts bootstrapping. However, both message shuffling networks and cryptotumblers are latency-tolerant and sometimes even deliberately stretch their operation over time to further increase the level of achieved privacy [ZMH⁺18]. If more timely service utilization is required, users can consider services valid for longer periods, thereby reducing the impact of the inflicted one-time overhead. In this case, users have to trade off delays against security, as longer validity periods devalue the protection offered by periodic PoW puzzles.

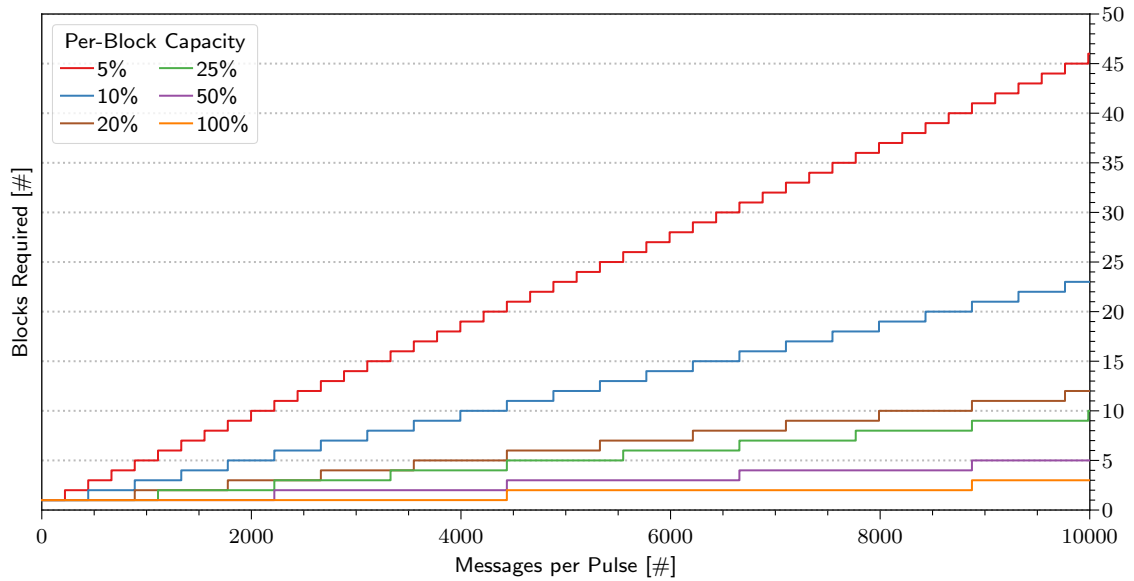


Figure 6.10 When using Bitcoin as its host blockchain, AnonBoot already scales to thousands of messages per pulse even for low per-block capacities, creating only a small storage footprint.

Contrarily, local peer selection only depends on individual user decisions and thus only relies on knowing a recent valid state of the peer repository, i.e., full synchronization up to the most recent pulse is desirable but not required. Namely, users can instantly sample privacy peers based on their current state, and thus AnonBoot preserves the low-latency requirement of anonymity networks, such as Tor.

Takeaway. We conclude that AnonBoot (a) has low synchronization overhead, (b) introduces feasible latencies for bootstrapping anonymity services, and (c) supports instant local peer selection.

6.8.2 Small Blockchain Footprint and Low Costs

We now show that AnonBoot realizes lightweight service discovery (**G2**) by assessing its impact on the host blockchain. Namely, we measure the footprint of AnonBoot, i.e., the additional storage requirements for Bitcoin full nodes due to AnonBoot’s coordination messages, using Bitcoin’s regression test mode. As we discussed in Section 6.4.1, AnonBoot can trade off its required share of the transaction bandwidth of the host blockchain during its pulses and the required length Δ_N of the negotiation phase to record all messages of the pulse via a configurable per-block capacity c . In the following, we assess how Δ_N must be chosen at the minimum to include an increasing number of AnonBoot messages for one pulse given different capacities c .

Figure 6.10 illustrates the required negotiation phase when reserving up to $c \cdot 100\%$ of a block’s available weight (cf. Section 2.3.4) with AnonBoot messages. On average, an `OP_RETURN` transaction for one peer advertisement or user request has a size of 307 Byte and a weight of 901 WU. Our results reveal that AnonBoot easily scales to large peer repositories and user bases with only a small footprint on Bitcoin.

When using a per-block capacity of only 10%, AnonBoot can support up to 10 000 messages during a negotiation phase with $L_N = 23$, i.e., a duration of roughly 4 h.

Peer repositories of size 1000, which are already sufficiently secure as we demonstrated in Section 6.7.2, have an only negligible impact on Bitcoin: Even when considering a small available per-block capacity of only 5% for AnonBoot messages, to account for Bitcoin’s low overall transaction throughput, the negotiation phase still concludes after $L_N = 5$ blocks with space for up to 109 service requests. We expect only a few service requests, as a single request suffices to bootstrap a shared anonymity service. The scalability then only depends on the used upper-layer protocol and is independent of AnonBoot.

Finally, we consider the costs inflicted by transaction fees when the operators of privacy peers and AnonBoot users publish their peer advertisements and service requests, respectively, to Bitcoin’s blockchain as the `OP_RETURN` payload of a transaction. For this purpose, we consider the same scenarios as for our proposal for minimum transaction fees to deter the insertion of blockchain content in Section 4.2.4. Namely, we distinguish a high-fee scenario with transaction fees of 269 Sat/Byte (Q4 of 2017) and a more recent low-fee scenario with fees of 4 Sat/Byte (Q3 of 2022). Furthermore, we consider the more recent Bitcoin market price of 21 198.80 USD/BTC from Q3 of 2022 (cf. Section 4.2.4). Under these circumstances, issuing AnonBoot messages was historically comparably expensive, with 17.51 USD per message in times of high transaction fees. However, in our recent scenario, these costs drop to 0.26 USD per message. We hence conclude that AnonBoot now has feasible costs for privacy peer operators. AnonBoot can further help amortize these costs by extending pulse lengths, e.g., to multiple days, while keeping short negotiation phases.

Takeaway. Overall, our analysis shows that AnonBoot can bootstrap over 100 shared anonymity services from a repository comprising 1000 privacy peers, while serving potentially thousands of users. Further, AnonBoot can scale well beyond this size with only a small storage impact on Bitcoin as its host blockchain and only low costs in terms of transaction fees for its participants.

6.9 Related Work

AnonBoot is related to previous works that tackle the *bootstrapping problem* and the risk of *Sybil attacks* in distributed systems, as well as proposals to *outsource* privacy services or the discovery of such services.

Bootstrapping Problem and Sybil Attacks. Both, the bootstrapping problem and the risk of Sybil attacks, are inherent to distributed protocols. In 2007, Knoll et al. [KWSW07] surveyed different approaches to finding entry points for established peer-to-peer (P2P) networks. Among other approaches, the authors proposed to bootstrap nodes through a distributed host system such as IRC [KWSW07]. Orthogonally, Levine et al. [LSM06] reviewed approaches to mitigate Sybil attacks. From their taxonomy, only resource testing and recurring costs and fees are applicable to fully decentralized systems without further assumptions. Recurring costs, namely

periodic PoW-based refreshments of eligibility, are a familiar building block in the field of blockchain sharding [LNZ⁺16, ZMR18, KJG⁺18], where recurring verification tasks are periodically redistributed among full nodes to improve the blockchain system’s scalability. AnonBoot adapts this Sybil-resistant building block in the form of its peer advertisements to implement the novel application that is securely bootstrapping distributed anonymity services. We extend the considerations of Knoll et al. [KWSW07] and rely on an established permissionless blockchain, such as Bitcoin, as AnonBoot’s host medium and trust anchor. This choice allows us to massively reduce the coordination complexity in AnonBoot since permissionless blockchains already offer a distributed means to reach a consensus of state.

Outsourcing Privacy Services and Service Discovery. Lee et al. [LPP19] proposed that the user’s Internet service provider (ISP) could provide privacy services on their behalf, such as address hiding or VPN tunneling. This work is orthogonal to our approach, as we bootstrap services without relying on a dedicated central operator. In fact, AnonBoot can also help users to increase their privacy against the ISP itself. AnonBoot’s local peer selection from the peer repository allows establishing a decentralized directory service for anonymity networks such as Tor. Similar contributions were made by other works, such as NISAN [PRR09] or ShadowWalker [MB09]. However, while both proposals prevent adversarial bias, they do not feature AnonBoot’s protection against Sybil attacks. Furthermore, these approaches do not address the challenges of heterogeneous privacy peers, such as Tor nodes with different exit policies. AnonBoot introduces the capabilities in its peer advertisements specifically to overcome this shortcoming. While approaches to realize sticky data policies on how to handle user privacy [PCM11] are related to our specification of peer capabilities, even policies that are highly compressed by domain knowledge-aware compression schemes, such as CPPL [HHS⁺16], may exceed our space limitations, especially considering Bitcoin’s allowed `OP_RETURN` payloads. Although CPPL may facilitate simple peer capabilities, more complex instances, such as Tor relay descriptors, require manual capability abstractions.

6.10 Conclusion and Future Work

In this chapter, we investigated the potential of blockchain-backed data management to promote novel applications (**P4**, cf. Section 1.1.2). As one promising example to demonstrate this potential, we identified a previously missing bootstrapping infrastructure that can facilitate the establishment of truly decentralized anonymity services provided by independent privacy peers. To this end, we introduced AnonBoot as a corresponding bootstrapping medium that coordinates exclusively via a permissionless host blockchain (**P3**), thereby enabling all participants to derive a common system state by just monitoring the host blockchain for relevant messages. Using these coordination capabilities and periodic PoW puzzles for interested privacy peers, AnonBoot then maintains a Sybil-resistant peer repository from which users can either directly sample privacy peers or request the bootstrapping of a shared anonymity service to be used by multiple users simultaneously.

AnonBoot is thus capable of improving the decentralization of online service discovery (**P1**). On the one hand, AnonBoot further decentralizes service directories that currently rely on few trusted peers for anonymity networks, such as Tor. On the other hand, AnonBoot resolves the current risk of Sybil attacks when consulting distributed anonymity services that are provided by only a few privacy peers, such as message shuffling systems or distributed cryptocurrency tumblers. Orthogonally, we also highlighted that AnonBoot has the potential for integrating further services, such as other services based on secure multiparty computation, as well.

Our evaluation shows that permissionless blockchains indeed constitute a well-suited foundation for bootstrapping distributed services: AnonBoot can easily maintain a peer repository of 1000 privacy peers when operating on top of Bitcoin while creating only a low storage footprint on its blockchain (**P2**). From this peer repository, AnonBoot can then bootstrap distributed anonymity services for handling potentially thousands of users. These results show that even the limited capabilities offered by Bitcoin's `OP_RETURN` payloads hold currently untapped potential for fueling further applications (**Q3**, cf. Section 1.2).

Our work illustrates the utility of AnonBoot for improving online service discovery. However, future work should further investigate desirable configurations for AnonBoot's parameters, such as the choice of its underlying PoW scheme, the frequency of pulses, and the length and per-block capacities of the negotiation phase to optimize its resilience against adversaries while keeping its operation costs and its storage impact on the host blockchain feasible. On a higher level, AnonBoot indicates the potential for further, currently unidentified applications to benefit from blockchain-backed data management. Hence, future work should continue to investigate these potentials to better our understanding on the true capabilities and benefits of permissionless blockchains.

7

Conclusion

The success of blockchain technology, as popularized by Bitcoin, can be attributed to the characteristic properties it offers (cf. Section 2.1): (a) decentralization, (b) immutability, (c) public verifiability, (d) pseudonymity, and (e) script-based state updates. Together, these properties enable flexible interactions between mutually distrusting participants without the need for consulting an especially trusted third party. As a result, blockchains now support a plethora of data-management applications, ranging from digital notary services over healthcare to global supply chains.

On the flip side, precisely these characteristic properties also introduced new challenges for the blockchain-backed data management (cf. Section 1.1.2): Blockchain systems have become attractive targets for malicious actors (**P1**), especially in the permissionless setting where also unknown users are allowed to obtain and write to the blockchain. Furthermore, the blockchain's desired immutability, which only allows appending new data, leads to increasing data volumes even if data becomes obsolete over time (**P2**). Moreover, the decentralized operation of blockchain systems and the requirement for participants to reach consensus about the blockchain's state introduce challenges regarding the coordination of nodes and the updatability of agreed-upon rules (**P3**). Finally, these limitations blurred the true utility of blockchain technology beyond its initial hype, i.e., blockchain-related developments struggled to identify which applications truly benefit from relying on a blockchain (**P4**).

In this dissertation, we hence set out to demystify and adjust the promises of blockchain-based data management in the permissionless setting. Correspondingly, the main goal for this dissertation was to better understand the true capabilities of blockchain technology to support data-management tasks and to improve its applicability. We tackled this goal guided by our three research questions (cf. Section 1.2), namely (a) how to moderate the content stored on permissionless blockchains (**Q1**), (b) whether and how we can retrofit existing blockchain systems with new functionality (**Q2**), and (c) what novel applications are truly enabled when using permissionless blockchains for data management (**Q3**).

We addressed these research questions with the four contributions (**C1–C4**) we presented in this dissertation. In the following, we revisit our contributions and discuss how they help answer our research questions (Section 7.1). Afterward, we conclude this dissertation by identifying directions for future work (Section 7.2) and posing some final remarks (Section 7.3).

7.1 Contributions

The contributions of this dissertation serve to provide answers for our three research questions. In this section, we give a brief summary of each contribution and their main results, and we discuss their relation to our research questions (cf. Figure 1.2).

7.1.1 Systematic Analysis of Non-Financial Blockchain Content

Our first contribution (**C1**) motivated the need for moderation capabilities in blockchain systems (**Q1**). Our holistic analysis of blockchain content insertion unveiled available content insertion methods, discussed benefits and especially potential negative consequences of this practice, and empirically documented that the irrevocable insertion of blockchain content is a commonly used practice across the blockchains of Bitcoin Core as well as its related forks Bitcoin Cash and Bitcoin SV.

Our analyses suggest that already single instances of illegal blockchain content have the potential to create severe consequences for the operators of full nodes, who have to store and distribute the full blockchain history, including such content. While the vast majority of content we identified is arguably harmless, the roughly 1600 files we extracted from Bitcoin Core’s blockchain include content that clearly highlights the potential problems, such as the depiction of a nude young woman or the hundreds of links to alleged child pornography. Even though our measurements suggest that content insertion has since slowed down in Bitcoin Core, our results show that the practice has not faded away but merely shifted to other systems; after moving to Bitcoin Cash, most content insertion seemingly shifted to Bitcoin SV. This blockchain further experienced unprecedented degrees of content insertion due to the endorsement of larger content via large `OP_RETURN` payloads and the Bitcoin Data Protocol, which was misused soon thereafter to engrave child abuse imagery [BBC19].

Overall, our first contribution revisited a core problem of permissionless blockchains that was vaguely theorized but never thoroughly investigated before. To the best of our knowledge, our work constituted the first systematic analysis of blockchain content insertion. As such, our work provides a solid foundation for assessing the consequences of unfiltered content insertion and strongly underpins the need for incorporating moderation capabilities in permissionless blockchains.

7.1.2 Mitigation of Unwanted Blockchain Content

Our second contribution focuses on mitigating blockchain content insertion (**C2**) to address the need for moderation capabilities (**Q1**) we identified with our systematic

analysis (**C1**). Along with our criteria for ensuring that our proposed mitigation schemes are effective against blockchain content insertion (cf. Section 4.1.1.2), we especially focused on their deployability to assess whether established blockchain systems can be retrofitted to provide such protection (**Q2**).

Due to this focus, we first investigated which strategies blockchain systems could deploy with minimum changes to the established consensus rules that prevent unwanted content from being accepted into the blockchain. While full nodes can delete transactions containing offending, objectionable, or illicit content locally without any required coordination [FHBS19], this behavior can also create undesirable side effects regarding the public verifiability of a blockchain. Instead, our prevention strategies explore how naïvely agreeing on content-filtering rules, deploying mandatory minimum fees, or hardening on-chain addresses against address manipulation can deter the insertion of unwanted content. Our results show that content-filtering rules are likely to impose significant and repeated coordination overhead that should be avoided in the permissionless setting. Contrarily, mandatory minimum fees and hardened addresses can successfully reduce blockchain content insertion. However, we showed that the phenomenon cannot be prevented entirely and, thus, unwanted content must be expected to irrevocably enter a blockchain at some point.

The impossibility of full prevention motivated us to slightly relax our requirement of retrofittability and revisit redactable blockchains. Previous proposals here were mostly trust-based [AMVA17] and essentially relied on a logically centralized redactor or voting-based [DMT19], which introduces undesirable redaction delays. With RedactChain and its approach to periodically redistribute redaction capabilities among small randomly selected and mutually overseeing redaction juries, we achieve a fast and simultaneously transparent redaction process.

The required need for introducing moderation capabilities for the blockchain-based data management cannot be realized without any changes to established blockchain designs. Our results indicate that effectively counteracting blockchain content insertion in the permissionless setting, which requires a fast and trustworthy process, implies that larger sacrifices to the characteristic properties of blockchains are needed than previously thought. However, our work lays out a path for gradually transitioning to blockchain systems that feature increasingly stronger protection. Notably, the combination of both approaches has the potential to efficiently prevent most content from entering the blockchain in the first place and thereby reduce the efforts required to then redact the remaining content in a swift and trustworthy manner.

7.1.3 Retrofitting Blockchains with Pruning Capabilities

With our third contribution, we tackle growing data volumes as a second core challenge for the blockchain-based data management, where immutable blockchains only allow appending new data and blockchain sizes already exceed hundreds of gigabytes. As this challenge is especially pressing for long-running and popular blockchains, retrospective deployability (**Q2**) is a central requirement for addressing this challenge.

In accordance with this requirement, CoinPrune (**C3**) introduces a retrofittable block-pruning scheme that can be gradually deployed to established blockchain systems such as Bitcoin. CoinPrune nodes periodically derive shareable snapshots from their local UTXO set at predefined block heights, which they then can serve to newly joining nodes instead of the full blockchain. Joining nodes can therefore catch up with the current state by obtaining only the comparably small headerchain, a recent snapshot, and the chaintail of full blocks added after the snapshot was created. Related work had already proposed new blockchain designs that utilize state-based synchronization processes to reduce the storage and synchronization overhead inflicted by relying on full blockchain copies. However, CoinPrune's positive-only feedback on the correctness of snapshots established by its miner reaffirmations now enables also Bitcoin nodes to profit from a trustworthy lightweight synchronization.

Our evaluation shows a tremendous potential to reduce the storage requirements for all full nodes supporting CoinPrune as well as vastly reduced synchronization times for newly joining nodes. Furthermore, CoinPrune's coordinated pruning opens up the potential for further improvements. Namely, we highlight additional leverage over the contents engraved in Bitcoin's UTXO set (**Q1**) and the preservation of otherwise prunable application-level data (**Q3**) as additional features of CoinPrune.

In summary, CoinPrune's gradual deployability demonstrates a path for retrofitting new features to those blockchains that are in dire need of the corresponding improvements. Ensuring this extensibility is desirable over prematurely replacing long-running blockchain systems, as trust has to be established over time.

7.1.4 Blockchain-based Bootstrapping of Anonymity Services

Our fourth and final contribution, AnonBoot (**C4**), demonstrates that permissionless blockchains still hold untapped potential for innovative new applications (**Q3**). Namely, AnonBoot realizes a trustworthy bootstrapping process for distributed anonymity services on top of a permissionless blockchain. We identified that permissionless blockchains are well-suited building blocks for this previously lacking application thanks to their decentralization, immutability, and transparency.

Correspondingly, AnonBoot utilizes an established blockchain to maintain a Sybil-resistant repository of prospective privacy peers as well as a medium to bootstrap shared anonymity services. Privacy-aware users can make use of AnonBoot as a service directory for local peer selection, e.g., as required for creating circuits of onion routers, or for shared anonymity services such as message shuffling networks or cryptocurrency tumblers. Our evaluation shows that AnonBoot can use Bitcoin as its host blockchain while periodically leaving only a small storage footprint.

After experiencing an initial hype for potential applications, proponents of blockchain technology struggled to pinpoint the circumstances under which relying on a blockchain, especially a permissionless one, is truly beneficial. Further maturing the technology thus does not only involve addressing its current technical challenges, but also requires assessing its aptitude to fuel additional applications. With AnonBoot, we have shown that blockchains still hold potential as suitable building blocks for decentralized applications if their characteristic properties are seized properly.

7.2 Future Work

This dissertation critically investigated the aptitude of permissionless blockchains to support data management tasks. Our insights and proposed solutions already constitute important steps toward a more sustainable approach to blockchain-based data management. However, there are important additional challenges to maturing blockchains for this task that were beyond the scope of this dissertation. In the following, we identify and discuss three main areas for future work, each of which is closely tied to one of our research questions: *better understanding blockchain content* in more diverse settings, intensifying *standardization and cooperation efforts*, and the *further exploration of applications*.

Better Understanding of Blockchain Content. In Chapter 3, we extensively surveyed content insertion methods as well as potential consequences. However, since the irrevocable insertion of a single illicit document could already have a serious impact [BBC19], further investigation of this problem space is mandatory. On the one hand, we need to improve our understanding of not only possible legal consequences, but also establish a framework to communicate what consequences users and full node operators need to expect. On the other hand, better understanding the available content insertion methods, also in other blockchain systems, would help further improve the technical countermeasures to mitigate these consequences. Furthermore, our contributions only consider non-financial blockchain content in the context of cryptocurrencies, whereas different use cases require the storage of different kinds of data on a blockchain. For instance, blockchains have been used in the past to record provenance data of global supply chains [BPM⁺21]. Similarly, federated data ecosystems are currently emerging and blockchains may facilitate their data exchanges in the future [LPMW22]. These contexts differ fundamentally from cryptocurrencies, which are self-contained in that their verifiability only covers the transactions and their interlinks. Namely, the blockchain necessarily records data relating to blockchain-external objects or processes, which creates additional challenges. For instance, full nodes may be unable to validate the correctness of such data, i.e., users have to accept the possibility of reading false data from such blockchains or verify the data integrity via side channels [PAM⁺20]. Furthermore, domain-specific data requires further modeling regarding blockchain storage, e.g., to define whether and how such data might be prunable to reduce storage requirements and costs for node operators, and thereby improve the system’s longevity. These considerations were out of scope for this dissertation but are required to further generalize the applicability of blockchain-based data management.

Standardization and Cooperation Efforts. After Bitcoin started to gain traction, blockchain technology began to evolve rapidly. While this new technology quickly fueled innovation, no clear path forward had emerged at that point. As a result, blockchain systems still may suffer from suboptimal documentation; for instance, the Bitcoin Wiki still constitutes the main resource for technical documentation of Bitcoin. Likewise, blockchain technology would benefit from further standardization efforts [Wor20]. These standardization efforts should especially emphasize the extensibility of blockchain systems. Altering established blockchain systems has already proven challenging in the past due to both technical and political implications

of updating independent nodes in a decentralized system, e.g., when Bitcoin Core deployed Segregated Witnesses (cf. Section 2.3.4). While Bitcoin invites proposals for improvement in its BIPs, which have a goal comparable to the RFCs maintained by the IETF, a comparably stringent process for evolving Bitcoin has yet to emerge. However, such a clear process is a precondition for ensuring a sustainable maintainability of long-running permissionless blockchain systems. Similarly, the discourse about whether or not to deploy proposed improvements should be further streamlined. We specifically chose to center the solutions proposed in this dissertation around their deployability to increase their acceptance, but unifying the channels for proposing features and voting for them would constitute one step further to ensuring that blockchain systems can continue to evolve in the long run.

Further Exploration of Applications. Blockchain technology is subject to tensions between euphoria [Bmwi18, Gan18, Eu19] and skepticism [WG18, Bar19, Sch19]. Hence, future work should continue to critically question the applicability of blockchain technology and further explore the settings that benefit from its characteristic properties. In the past, we have already contributed to this exploration and investigated the particular benefits of blockchain technology in different applications, such as car insurance [BBMW18], licensing of education material [MPW20], and tracking and tracing in supply chains [PAM⁺20, BPM⁺21, WMP⁺22]. We anticipate that, while many applications do not require a blockchain-based solution, the potential of this building block is not yet exhausted.

7.3 Closing Remarks

This dissertation focused on the applicability of permissionless blockchains in data management systems. As part of our journey to better understand and remedy the implications of blockchain-based data management, we highlighted the semantics of content stored on the blockchain and growing data volumes as two core challenges. Overall, our contributions provide important steps to tackle these challenges by advocating for moderation in blockchain systems, focusing on the retrospective deployability of new features to benefit long-running systems with established reputation, and demonstrating that the technology still has an untapped potential.

Our insights and approaches sometimes challenged established views within parts of the blockchain community. For instance, our assessment regarding illicit blockchain content has been theorized but also downplayed before. We hence appreciate that our work and the attention it received helped reignite this important discussion, even though it was partially transported in an oversimplified manner. Further, our contributions provide valuable insights into the trade-off between retrospectively altering established blockchains for additional protection or features and the retention of the characteristic properties of permissionless blockchains.

We are confident that the contributions of this dissertation serve as a suitable foundation for future research. However, we focused only on a few of many challenges of permissionless blockchains and their data management. Thus, we are looking forward to future endeavors further improving blockchain technology beyond its hype.

Abbreviations and Acronyms

ASCII	American Standard Code for Information Interchange	82
AVL	Adelson-Velsky and Landis tree	164
BCH	Bitcoin Cash	11
BDP	Bitcoin Data Protocol	73
BIP	Bitcoin Improvement Proposal	17
BSV	Bitcoin SV	2
BTC	Bitcoin Core	14
CHET	chameleon-hash with ephemeral trapdoor	128
CHF	chameleon hash function	125
CP-ABE	ciphertext-policy attribute-based encryption	128
CPU	central processing unit	206
CRC32	32-bit cyclic redundancy code	72
CSV	Comma-Separated Values	98
DER	Distinguished Encoding Rules	32
DHT	distributed hash table	15
DKG	distributed key generation	125
DNS	Domain Name System	55
DoS	denial of service	5
DRM	digital rights management	74
DVD	digital video disc	96
EC2	Elastic Compute Cloud	156
ECC	elliptic curve cryptography	32
ECDHC-CDE	Elliptic Curve Diffie-Hellman Chain-based Covert Data Embedding	69
ECDSA	Elliptic Curve Digital Signature Algorithm	32
FPLE	functionality-preserving local erasure	110
FPR	false-positive rate	113
GDPR	General Data Protection Regulation	77
GPL	GNU General Public License	125
HC-CDE	Hash Chain-based Covert Data Embedding	69
HTML	Hypertext Markup Language	98
IETF	Internet Engineering Task Force	220
IP	Internet Protocol	56
IPv4	Internet Protocol version 4	196
IPv6	Internet Protocol version 6	196
IRC	Internet Relay Chat	55
ISP	Internet service provider	213

IWF Internet Watch Foundation	138
JSON JavaScript Object Notation	83
LSB least significant bit	69
mempool memory pool	20
MIME Multipurpose Internet Mail Extensions	73
MIT Massachusetts Institute of Technology	159
nonce number used once	26
P2MS Pay to Multi-Signature	40
P2P peer-to-peer	1
P2PK Pay to Public Key	34
P2PKC Pay to Public-Key Commitment	122
P2PKH Pay to Public-Key Hash	34
P2SC Pay to Script Commitment	122
P2SH Pay to Script Hash	40
P2WPKH Pay to Witness Public-Key Hash	41
P2WSH Pay to Witness Script Hash	41
P2X Pay to X	42
PC personal computer	178
PDF Portable Document Format	62
PoS Proof of Stake	206
PoW Proof of Work	4
PRNG pseudo-random number generator	143
RAM Random Access Memory	148
RFC Request for Comments	220
RPC remote procedure call	82
RSA Rivest-Shamir-Adleman cryptosystem	32
SEC Standards for Efficient Cryptography	32
SegWit Segregated Witness	25
SMC secure multiparty computation	134
SPV simple payment verification	29
StGB German Criminal Code (Strafgesetzbuch)	80
TCP Transmission Control Protocol	148
URL Uniform Resource Locator	14
UTXO unspent transaction output	29
VPN virtual private network	213
WTO World Trade Organization	2
XML Extensible Markup Language	98
ZKP zero-knowledge proof	23

Bibliography

- [ABC⁺03] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, dec 2003.
- [ABL⁺18] Divesh Aggarwal, Gavin Brennen, Troy Lee, Miklos Santha, and Marco Tomamichel. Quantum Attacks on Bitcoin, and How to Protect Against Them. *Ledger*, 3, Oct. 2018.
- [AC22] Thiago Leucz Astrizi and Ricardo Custódio. Redacting Blockchain Without Exposing Chameleon Hash Collisions. In Alastair R. Beresford, Arpita Patra, and Emanuele Bellini, editors, *Cryptology and Network Security (CANS)*, pages 339–358. Springer Cham, 2022.
- [AD01] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM)*, pages 310–317. ACM, 2001.
- [AFKU20] Erikson Júlio De Aguiar, Bruno S. Faiçal, Bhaskar Krishnamachari, and Jô Ueyama. A Survey of Blockchain-Based Strategies for Healthcare. *ACM Computing Surveys*, 53(2), March 2020.
- [Ahl20] Vincent Ahlrichs. Design and Implementation of Flexible and Trustworthy Redactions for Permissionless Blockchains. Master’s thesis, RWTH Aachen University, June 2020.
- [AM05] Giuseppe Ateniese and Breno de Medeiros. On the Key Exposure Problem in Chameleon Hashes. In Carlo Blundo and Stelvio Cimato, editors, *Security in Communication Networks*, pages 165–179. Springer Berlin Heidelberg, 2005.
- [AMLH15] Syed Taha Ali, Patrick McCorry, Peter Hyun-Jeen Lee, and Feng Hao. ZombieCoin: Powering Next-Generation Botnets with Bitcoin. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, pages 34–48. Springer Berlin Heidelberg, 2015.
- [AMVA17] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton Andrade. Redactable Blockchain – or – Rewriting History in Bitcoin and Friends. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 111–126. IEEE, 2017.

- [And11] Gavin Andresen. BIP 11: M-of-N Standard Transactions, 2011. URL: https://web.archive.org/web/20220119154035if_/https://github.com/bitcoin/bips/blob/master/bip-0011.mediawiki (archived on 2022-01-19).
- [And12a] Gavin Andresen. BIP 16: Pay to Script Hash, 2012. URL: https://web.archive.org/web/20220110125000if_/https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki (archived on 2022-01-10).
- [And12b] Gavin Andresen. BIP 34: Block v2, Height in Coinbase, 2012. URL: https://web.archive.org/web/20220110125647if_/https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki (archived on 2022-01-10).
- [And13] Gavin Andresen. BIP 50: March 2013 Chain Fork Post-Mortem, 2013. URL: https://web.archive.org/web/20211004145138if_/https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki (archived on 2021-10-04).
- [Ano17] Andreas M. Anotnopoulos. *Mastering Bitcoin*. O'Reilly, second edition, 2017.
- [ASL19] Kondapally Ashritha, M. Sindhu, and K.V. Lakshmy. Redactable Blockchain using Enhanced Chameleon Hash Function. In *5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, pages 323–328. IEEE, 2019.
- [ASN17] Muneeb Ali, Ryan Shea, Jude Nelson, and Michael J. Freedman. Blockstack: A New Internet for Decentralized Applications. White paper, 2017, URL: <https://pdos.csail.mit.edu/6.824/papers/blockstack-2017.pdf>.
- [AW19] Hany F. Atlam and Gary B. Wills. Technical aspects of blockchain and IoT. In Shihoh Kim, Ganesh Chandra Deka, and Peng Zhang, editors, *Role of Blockchain Technology in IoT Applications*, volume 115 of *Advances in Computers*, chapter 1, pages 1–39. Elsevier, 2019.
- [Bad14] Badbitcoin.org. The Definitive Bitcoin and Cryptocurrency Fraud List, 2014. URL: https://web.archive.org/web/20230221122927id_/https://www.badbitcoin.org (archived on 2023-02-21).
- [Bar19] Gregory Barber. What's Blockchain Actually Good for, Anyway? For Now, Not Much, 2019. URL: https://web.archive.org/web/20211028094619if_/https://www.wired.com/story/whats-blockchain-good-for-not-much (archived on 2021-10-28).
- [BBC18] BBC News. 'Child porn links could make Bitcoin blockchain illegal', 2018. URL: https://web.archive.org/web/20220505110717if_/https://www.bbc.com/news/technology-43485572 (archived on 2022-05-05).
- [BBC19] BBC News. Child abuse images hidden in crypto-currency blockchain, 2019. URL: <https://web.archive.org/web/20210625141025/https://www.bbc.com/news/technology-47130268> (archived on 2021-06-25).
- [BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology (CRYPTO)*, pages 561–586. Springer Cham, 2019.

- [BBMW18] Lennart Bader, Jens Christoph Bürger, Roman Matzutt, and Klaus Wehrle. Smart Contract-based Car Insurance Policies. In *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2018.
- [BBP19] Massimo Bartoletti, Bryn Bellomy, and Livio Pompianu. A Journey into Bitcoin Metadata. *Journal of Grid Computing*, 17(1):3–22, 2019.
- [BBSU12] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to Better — How to Make Bitcoin a Better Currency. In Angelos D. Keromytis, editor, *Financial Cryptography and Data Security (FC)*, pages 399–414. Springer Berlin Heidelberg, 2012.
- [BCG15] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On Bitcoin as a public randomness source. Cryptology ePrint Archive, Paper 2015/1015, 2015. URL: <https://eprint.iacr.org/2015/1015>.
- [BEHE19] Thomas Buocz, Tina Ehrke-Rabel, Elisabeth Hödl, and Iris Eisenberger. Bitcoin and the GDPR: Allocating responsibility in distributed networks. *Computer Law & Security Review*, 35(2):182–198, 2019.
- [BHH⁺14] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic Curve Cryptography in Practice. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security (FC)*, pages 157–175. Springer Berlin Heidelberg, 2014.
- [BIH12] John B. Bellinger III and Murad Hussain. Freedom of Speech: The Great Divide and the Common Ground between the United States and the Rest of the World. In *Islamic Law and International Human Rights Law: Searching for Common Ground?*, pages 168–180. 2012.
- [Bit10a] Bitcoin Project. Block – Bitcoin Wiki, 2010. URL: https://web.archive.org/web/20220110114832id_/https://en.bitcoin.it/wiki/Block (archived on 2022-01-10).
- [Bit10b] Bitcoin Project. Network, 2010. URL: https://web.archive.org/web/20220420084226id_/https://en.bitcoin.it/wiki/Network (archived on 2022-04-20).
- [Bit10c] Bitcoin Project. Protocol documentation – Bitcoin Wiki, 2010. URL: https://web.archive.org/web/20220109215004id_/https://en.bitcoin.it/wiki/Protocol_documentation (archived on 2022-01-09).
- [Bit10d] Bitcoin Project. Script – Bitcoin Wiki, 2010. URL: https://web.archive.org/web/20210716142939id_/https://en.bitcoin.it/wiki/Script (archived on 2021-07-16).
- [Bit10e] Bitcoin Project. Target – Bitcoin Wiki, 2010. URL: https://web.archive.org/web/20220110120049id_/https://en.bitcoin.it/wiki/Target (archived on 2022-01-10).
- [Bit10f] Bitcoin Project. Transaction – Bitcoin Wiki, 2010. URL: https://web.archive.org/web/20220119153835id_/https://en.bitcoin.it/wiki/Transaction (archived on 2022-01-19).

- [Bit10g] Bitcoin Project. Weaknesses, 2010. URL: https://web.archive.org/web/20220505093610id_/https://en.bitcoin.it/wiki/Weaknesses (archived on 2022-05-05).
- [Bit11a] Bitcoin Project. Base58Check encoding – Bitcoin Wiki, 2011. URL: https://web.archive.org/web/20220110115217id_/https://en.bitcoin.it/wiki/Base58Check_encoding (archived on 2022-01-10).
- [Bit11b] Bitcoin Project. Block timestamp – Bitcoin Wiki, 2011. URL: https://web.archive.org/web/20220110120431id_/https://en.bitcoin.it/wiki/Block_timestamp (archived on 2022-01-10).
- [Bit11c] Bitcoin Project. Contract – Bitcoin Wiki, 2011. URL: https://web.archive.org/web/20220114115058id_/https://en.bitcoin.it/wiki/Contract (archived on 2022-01-14).
- [Bit11d] Bitcoin Project. Protocol rules – Bitcoin Wiki, 2011. URL: https://web.archive.org/web/20210714145539id_/https://en.bitcoin.it/wiki/Protocol_rules (archived on 2021-07-14).
- [Bit11e] Bitcoin Project. Secp256k1 – Bitcoin Wiki, 2011. URL: https://web.archive.org/web/20220106105626id_/https://en.bitcoin.it/wiki/Secp256k1 (archived on 2022-01-06).
- [Bit11f] Bitcoin Project. Technical background of version 1 Bitcoin addresses – Bitcoin Wiki, 2011. URL: https://web.archive.org/web/20211211113118id_/https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses (archived on 2022-01-11).
- [Bit11g] Bitcoin Project. Wallet – Bitcoin Wiki, 2011. URL: https://web.archive.org/web/20220110121739id_/https://en.bitcoin.it/wiki/Wallet (archived on 2022-01-10).
- [Bit12a] Bitcoin Project. Hot wallet – Bitcoin Wiki, 2012. URL: https://web.archive.org/web/20220110121500id_/https://en.bitcoin.it/wiki/Hot_wallet (archived on 2022-01-10).
- [Bit12b] Bitcoin Project. List of address prefixes – Bitcoin Wiki, 2012. URL: https://web.archive.org/web/20220119153616id_/https://en.bitcoin.it/wiki/List_of_address_prefixes (archived on 2022-01-19).
- [Bit12c] Bitcoin Project. Vanitygen – Bitcoin Wiki, 2012. URL: https://web.archive.org/web/20211218231904id_/https://en.bitcoin.it/wiki/Vanitygen (archived on 2022-01-10).
- [Bit13a] Bitcoin Project. Bitcoin-Qt version 0.8.0 released, 2013. URL: https://web.archive.org/web/20230107151517id_/https://bitcoin.org/en/release/v0.8.0 (archived on 2023-01-07).
- [Bit13b] Bitcoin Project. Mining hardware comparison, 2013. URL: https://web.archive.org/web/20230213063251id_/https://en.bitcoin.it/wiki/Mining_hardware_comparison (archived on 2023-02-13).
- [Bit13c] Bitnodes. Bitcoin Network 8 Years Charts, 2013. URL: <https://bitnodes.io/dashboard/8y>.

- [Bit14a] Bitcoin Project. Bitcoin Core version 0.9.0 released, 2014. URL: https://web.archive.org/web/20220117144044id_/https://bitcoin.org/en/release/v0.9.0 (archived on 2022-01-17).
- [Bit14b] Bitcoin Project. Colored Coins – Bitcoin Wiki, 2014. URL: https://web.archive.org/web/20210719085835id_/https://en.bitcoin.it/wiki/Colored_Coins (archived on 2021-07-19).
- [Bit14c] Bitcoin Project. Full node, 2014. URL: https://web.archive.org/web/20220420125811id_/https://en.bitcoin.it/wiki/Full_node (archived on 2022-04-20).
- [Bit14d] Bitcoin Project. Hardfork, 2014. URL: https://web.archive.org/web/20220412073434id_/https://en.bitcoin.it/wiki/Hardfork (archived on 2022-04-12).
- [Bit14e] Bitcoin Project. Softfork, 2014. URL: https://web.archive.org/web/20220412073735id_/https://en.bitcoin.it/wiki/Softfork (archived on 2022-04-12).
- [Bit15a] Bitcoin Project. Bitcoin Core version 0.10.0 released, 2015. URL: https://web.archive.org/web/20220429195953id_/https://bitcoin.org/en/release/v0.10.0 (archived on 2022-04-29).
- [Bit15b] Bitcoin Project. Bitcoin Core version 0.11.0 released, 2015. URL: https://web.archive.org/web/20201218153751id_/https://bitcoin.org/en/release/v0.11.0 (archived on 2020-12-18).
- [Bit15c] Bitcoin Project. Block size limit controversy – Bitcoin Wiki, 2015. URL: https://web.archive.org/web/20220110123359id_/https://en.bitcoin.it/wiki/Block_size_limit_controversy (archived on 2022-01-10).
- [Bit15d] Bitcoin Project. Running a Full Node, 2015. URL: https://web.archive.org/web/20230104145918if_/https://bitcoin.org/en/full-node (archived on 2023-01-04).
- [Bit16a] Bitcoin Project. Bitcoin Core version 0.12.0 released, 2016. URL: https://web.archive.org/web/20220121101519id_/https://bitcoin.org/en/release/v0.12.0 (archived on 2021-01-21).
- [Bit16b] Bitcoin Project. Bitcoin Core version 0.13.0 released, 2016. URL: https://web.archive.org/web/20230107152642id_/https://bitcoin.org/en/release/v0.13.0 (archived on 2023-01-07).
- [Bit16c] Bitcoin Project. Quantum computing and Bitcoin – Bitcoin Wiki, 2016. URL: https://web.archive.org/web/20220110165940id_/https://en.bitcoin.it/wiki/Quantum_computing_and_Bitcoin (archived on 2022-01-10).
- [Bit17a] Bitcoin Project. Bitcoin Core version 0.14.0 released, 2017. URL: https://web.archive.org/web/20230107153706id_/https://bitcoin.org/en/release/v0.14.0 (archived on 2023-01-07).
- [Bit17b] Bitcoin Project. Bitcoin Core version 0.15.0 released, 2017. URL: https://web.archive.org/web/20210516131005id_/https://bitcoin.org/en/release/v0.15.0 (archived on 2021-05-16).

- [Bit17c] Bitcoin Project. Segregated Witness – Bitcoin Wiki, 2017. URL: https://web.archive.org/web/20220110123550id_/https://en.bitcoin.it/wiki/Segregated_Witness (archived on 2022-01-10).
- [Bit17d] Bitcoin Project. Storing bitcoins – Bitcoin Wiki, 2017. URL: https://web.archive.org/web/20220110122341id_/https://en.bitcoin.it/wiki/Storing_bitcoins (archived on 2022-01-10).
- [Bit17e] Bitcoincash.org. Bitcoin Cash - Peer-to-Peer Electronic Cash, 2017. URL: https://web.archive.org/web/20220411121222js_/https://bitcoincash.org (archived on 2022-04-11).
- [Bit18a] Bitcoin Association. Bitcoin SV is the Original Bitcoin, 2018. URL: https://web.archive.org/web/20210806101956id_/https://bitcoinsv.com/en (archived on 2021-08-06).
- [Bit18b] Bitcoin Association. History of Bitcoin, 2018. URL: https://web.archive.org/web/20210930095421id_/https://bitcoinsv.com/en/learn/history-of-bitcoin (archived on 2021-09-30).
- [Bit18c] Bitcoin Association. Protocol Comparison, 2018. URL: <https://web.archive.org/web/20210814202551/https://bitcoinsv.com/en/learn/protocol-comparison> (archived on 2021-08-14).
- [Bit18d] Bitcoin Project. Lightweight Node, 2018. URL: https://web.archive.org/web/20211212174441id_/https://en.bitcoin.it/wiki/Lightweight_node (archived on 2021-12-12).
- [Bit21a] Bitcoin Project. Bitcoin Core file system, 2021. URL: https://web.archive.org/web/20220426124305if_/https://github.com/bitcoin/bitcoin/blob/v22.0/doc/files.md (archived on 2022-04-26).
- [Bit21b] Bitcoin Project. OP_CHECKMULTISIG – Bitcoin Wiki, 2021. URL: https://web.archive.org/web/20220119145232id_/https://en.bitcoin.it/wiki/OP_CHECKMULTISIG (archived on 2022-01-19).
- [Bit22] Bitcoin Project. Bitcoin Core 22.0, 2022. URL: https://web.archive.org/web/20220118084645id_/https://bitcoin.org/en/releases/22.0 (archived on 2022-01-18).
- [BK17] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. *Ledger*, 2:1–30, 4 2017.
- [BKLZ20] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. FlyClient: Super-Light Clients for Cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (S&P)*, pages 928–946. IEEE, 2020.
- [Blo11a] Blockchain.com. Blockchain Charts – Blockchain Size (MB), 2011. URL: <https://www.blockchain.com/charts/blocks-size> (accessed on 2023-01-04).
- [Blo11b] Blockchain.com. Blockchain Charts – Market Price (USD), 2011. URL: https://web.archive.org/web/20221025151248js_/https://www.blockchain.com/explorer/charts/market-price (archived on 2022-11-25).

- [BMC⁺15] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Symposium on Security and Privacy (S&P)*, pages 104–121. IEEE, 2015.
- [Bmwi18] German Federal Ministry for Economic Affairs and Energy and German Federal Ministry of Finance. Blockchain Strategy of the Federal Government – We Set Out the Course for the Token Economy. Federal strategy report, 2018, URL: <https://www.bmwi.de/Redaktion/EN/Publikationen/Digitale-Welt/blockchain-strategy.html>.
- [BNM⁺14] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for Bitcoin with Accountable Mixes. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security (FC)*, pages 486–504. Springer Berlin Heidelberg, 2014.
- [BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10. ACM, 1988.
- [BP15] Alex Biryukov and Ivan Pustogarov. Bitcoin over Tor isn’t a Good Idea. In *Symposium on Security and Privacy (S&P)*, pages 122–134. IEEE, 2015.
- [BP17] Massimo Bartoletti and Livio Pompianu. An Analysis of Bitcoin OP_RETURN Metadata. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security (FC)*, pages 218–230. Springer Cham, 2017.
- [BPM⁺21] Lennart Bader, Jan Pennekamp, Roman Matzutt, David Hedderich, Markus Kowalski, Volker Lücken, and Klaus Wehrle. Blockchain-Based Privacy Preservation for Supply Chains Supporting Lightweight Multi-Hop Information Accountability. *Information Processing & Management*, 58(3), May 2021.
- [Bra84] Gabriel Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$ -resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 154–162. ACM, 1984.
- [Bro09] Daniel R. L. Brown. SEC 1: Elliptic Curve Cryptography. Standard, 2009, URL: <https://secg.org/sec1-v2.pdf>.
- [Bro10a] Christian M. Bron. Germany–Judicial and Legislative Developments on Internet Child Pornography. Technical report, 2010. Accessed 2020-10-19.
- [Bro10b] Daniel R. L. Brown. SEC 2: Recommended Elliptic Curve Domain Parameters. White paper, 2010, URL: <https://www.secg.org/sec2-v2.pdf>.
- [Bru14] J. D. Bruce. The Mini-Blockchain Scheme. White paper, 2014, URL: <http://cryptonite.info/files/mbc-scheme-rev3.pdf>.

- [BSCG⁺14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *Symposium on Security and Privacy (S&P)*, pages 459–474. IEEE, 2014.
- [BSI19] Federal Office for Information Security (BSI). Towards Secure Blockchains. Technical report, 2019, URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Secure_Blockchain.pdf.
- [BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *Symposium on Security and Privacy (S&P)*, pages 321–334. IEEE, 2007.
- [But13] Vitalik Buterin. Ethereum Whitepaper. White paper, 2013, URL: https://web.archive.org/web/20210609221529id_/https://ethereum.org/en/whitepaper (archived on 2021-06-09).
- [CDE⁺16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On Scaling Decentralized Blockchains. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security (FC)*, pages 106–125. Springer Berlin Heidelberg, 2016.
- [CDK⁺17] Jan Camenisch, David Derler, Stephan Krenn, Henrich C. Pöhls, Kai Samelin, and Daniel Slamanig. Chameleon-Hashes with Ephemeral Trapdoors. In Serge Fehr, editor, *Public-Key Cryptography (PKC)*, pages 152–182. Springer Berlin Heidelberg, 2017.
- [Cha81] David L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.
- [Cha13] Flavien Charlon. Open Assets Protocol, 2013. URL: https://web.archive.org/web/20220509115650if_/https://github.com/OpenAssets/open-assets-protocol (archived on 2022-05-09).
- [Che16] Maxim Chesnokow. CryptoGraffiti: Permanently Preserve Images on the Blockchain, 2016. URL: https://web.archive.org/web/20220512075104if_/https://news.bitcoin.com/cryptograffiti-images-blockchain (archived on 2022-05-12).
- [CLO16] Alexander Chepurnoy, Mario Larangeira, and Alexander Ojiganov. Rollerchain, a Blockchain With Safely Pruneable Full Blocks, 2016. URL: <https://arxiv.org/abs/1603.07926>. arXiv:1603.07926.
- [CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.
- [Coi13] CoinMarketCap. Global Cryptocurrency Charts, 2013. URL: <https://coinmarketcap.com/charts/#dominance-percentage>.
- [Com15] Committee to Protect Journalists. Chinese journalist accused of illegally acquiring state secrets, 2015. URL: https://web.archive.org/web/20220516061016if_/https://cpj.org/2015/10/chinese-journalist-accused-of-illegally-acquiring (archived on 2022-05-16).

- [CPPK11] Sambuddho Chakravarty, Georgios Portokalidis, Michalis Polychronakis, and Angelos D. Keromytis. Detecting Traffic Snooping in Tor Using Decoys. In Robin Sommer, Davide Balzarotti, and Gregor Maier, editors, *Recent Advances in Intrusion Detection*, pages 222–241. Springer Berlin Heidelberg, 2011.
- [CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, pages 46–66. Springer Berlin Heidelberg, 2001.
- [CTDR19] Kirsten Cremona, Donald Tabone, and Clifford De Raffaele. Cybersecurity and the Blockchain: Preventing the Insertion of Child Pornography Images. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 197–204. IEEE, 2019.
- [CW20] Huan Chen and Yijie Wang. MiniChain: A lightweight protocol to combat the UTXO growth in public blockchain. *Journal of Parallel and Distributed Computing*, 143:67–76, 2020.
- [CYG⁺22] Haotian Cao, Hao Yin, Feng Gao, Zijian Zhang, Bakh Khoussainov, Shubin Xu, and Liehuang Zhu. Chain-based Covert Data Embedding Schemes in Blockchain. *Internet of Things Journal*, 9(16):14699–14707, 2022.
- [DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption*, pages 71–82. Springer Berlin Heidelberg, 1996.
- [DCCD⁺19] Claudio Di Ciccio, Alessio Cecconi, Marlon Dumas, Luciano García-Bañuelos, Orlenys López-Pintado, Qinghua Lu, Jan Mendling, Alexander Ponomarev, An Binh Tran, and Ingo Weber. Blockchain Support for Collaborative Business Processes. *Informatik Spektrum*, 42(3):182–190, 2019.
- [DDM03] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: design of a type III anonymous remailer protocol. In *Symposium on Security and Privacy (S&P)*, pages 20–15. IEEE, 2003.
- [DDN⁺22] Abi Djanogly, Yaniv Dudu, Andrew Newman, Eric Wolkstein, and Dana Yosifovich. The State of Consumer Cybersecurity 2022. White paper, 2022, URL: <https://reasonlabs.com/research/cybersecurity-report>.
- [DKJ17] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Towards an Optimized BlockChain for IoT. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 173–178. IEEE/ACM, 2017.
- [DKJ19] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. MOF-BC: A memory optimized and flexible blockchain for large scale networks. *Future Generation Computer Systems*, 92:357–373, 2019.
- [DM19] Roger Dingledine and Nick Mathewson. Tor Protocol Specification, 2019. URL: https://web.archive.org/web/20230303115148id_/https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt (archived on 2023-03-03).

- [DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*. USENIX Association, 2004.
- [DMT19] Dominic Deuber, Bernardo Magri, and Sri Aravinda Krishnan Thyagarajan. Redactable Blockchain in the Permissionless Setting. In *2019 IEEE Symposium on Security and Privacy (S&P)*, pages 124–138. IEEE, 2019.
- [Dol12] Stephen Dolan. jq, 2012. URL: https://web.archive.org/web/20220726203355if_/https://stedolan.github.io/jq (archived on 2022-07-26).
- [Dou16] David M. Douglas. Doxing: a conceptual analysis. *Ethics and Information Technology*, 18(3):199–210, 2016.
- [DPNH19] Sergi Delgado-Segura, Cristina Pérez-Solà, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. Analysis of the Bitcoin UTXO Set. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *Financial Cryptography and Data Security Workshops (FC Workshops)*, pages 78–91. Springer Berlin Heidelberg, 2019.
- [DSSS19] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. In *26th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2019.
- [DTF19] Mehmet Demir, Ozgur Turetken, and Alexander Ferworn. Blockchain Based Transparent Vehicle Insurance Management. In *2019 Sixth International Conference on Software Defined Systems (SDS)*, pages 213–220. IEEE, 2019.
- [DVP⁺02] Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, pages 207–216. ACM, 2002.
- [DW13] Christian Decker and Roger Wattenhofer. Information Propagation in the Bitcoin Network. In *2013 International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2013.
- [DZZ19] Hong-Ning Dai, Zibin Zheng, and Yan Zhang. Blockchain for Internet of Things: A Survey. *IEEE Internet of Things Journal*, 6(5):8076–8094, 2019.
- [Ers16] Erich Erstu. CryptoGraffiti Client v2.0, 2016. URL: https://web.archive.org/web/20220512075644if_/https://github.com/1Hyena/cryptograffiti (archived on 2022-05-12).
- [ES14] Ittay Eyal and Emin Gün Sirer. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security (FC)*, pages 436–454. Springer Berlin Heidelberg, 2014.

- [Eth22] The Ethereum Foundation. Ethash, 2022. URL: https://web.archive.org/web/20230226233421if_/https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/mining-algorithms/ethash (archived on 2023-02-26).
- [Eu19] Susana Figueiredo do Nascimento, Alexandre Roque Mendes Polvora, Amanda Anderberg, Elena Andonova, Mario Bellia, Ludovic Calés, Andreia Inamorato dos Santos, Ioannis Kounelis, Igor Nai Fovino, Marco Petracco Giudici, Evangelia Papanagiotou, Maciej Sobolewski, Fiammetta Rossetti, and Laura Spirito. Blockchain Now and Tomorrow. Eur 29813 en, scientific and technical research reports, 2019, URL: <https://publications.jrc.ec.europa.eu/repository/handle/JRC117255>.
- [FCS18] Tooba Faisal, Nicolas Courtois, and Antoaneta Serguieva. The Evolution of Embedding Metadata in Blockchain Transactions. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018.
- [Fel87] Paul Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (SFCS)*, pages 427–438. ACM, 1987.
- [FHBS19] Martin Florian, Sebastian Henningsen, Sophie Beaucamp, and Björn Scheuermann. Erasing Data from Blockchain Nodes. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 367–376. IEEE, 2019.
- [FKS15] Michael Fleder, Michael Kester, and Pillai Sudeep. Bitcoin Transaction Graph Analysis, 2015. URL: <https://arxiv.org/abs/1502.01657>. arXiv:502.01657.
- [FMR16] Damiano Di Francesco Maesa, Andrea Marino, and Laura Ricci. Uncovering the Bitcoin Blockchain: An Analysis of the Full Users Graph. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 537–546. IEEE, 2016.
- [Gan18] Emanuelle Ganne. *Can Blockchain revolutionize international trade?* WTO Publications, Geneva, Switzerland, 2018.
- [GC17] Alexander Grech and Anthony F. Camilleri. *Blockchain in Education*. Publications Office of the European Union, Luxembourg, 2017.
- [GDPR] The European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), 2016. URL: <https://eur-lex.europa.eu/legal-content/en/TXT/?uri=CELEX:02016R0679-20160504>.
- [Gen19] Matthias Geniar. Initial impressions on running a Bitcoin Core full node, 2019. URL: <https://web.archive.org/web/20210202105455/https://matthias.be/initial-impressions-on-running-a-bitcoin-core-full-node> (archived on 2021-02-02).

- [GFGC21] Mar Gimenez-Aguilar, Jose M. De Fuentes, Lorena González-Manzano, and Carmen Camara. Zephyrus: An Information Hiding Mechanism Leveraging Ethereum Data Fields. *IEEE Access*, 9:118553–118570, 2021.
- [Gib18] Samuel Gibbs. Child abuse imagery found within bitcoin’s blockchain, 2018. URL: https://web.archive.org/web/20220505110325if_/https://www.theguardian.com/technology/2018/mar/20/child-abuse-imagery-bitcoin-blockchain-illegal-content (archived on 2022-05-05).
- [GJA03] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A Reputation System for Peer-to-Peer Networks. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 144–152. ACM, 2003.
- [GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.
- [GM17] Matthew Green and Ian Miers. Bolt: Anonymous Payment Channels for Decentralized Currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 473–489. ACM, 2017.
- [GMC20] Alexandre Augusto Giron, Jean Everson Martina, and Ricardo Custódio. Bitcoin Blockchain Steganographic Analysis. In Jianying Zhou, Mauro Conti, Chuadhry Mujeeb Ahmed, Man Ho Au, Lejla Batina, Zhou Li, Jingqiang Lin, Eleonora Losiouk, Bo Luo, Suryadipta Majumdar, Weizhi Meng, Martín Ochoa, Stjepan Picek, Georgios Portokalidis, Cong Wang, and Kehuan Zhang, editors, *Applied Cryptography and Network Security Workshops (ACNS Workshops)*, pages 41–57. Springer Cham, 2020.
- [GMC21] Alexandre Augusto Giron, Jean Everson Martina, and Ricardo Custódio. Steganographic Analysis of Blockchains. *Sensors*, 21(12), 2021.
- [GMF22] Marcel Gregoriadis, Robert Muth, and Martin Florian. Analysis of Arbitrary Content on Blockchain-Based Systems Using BigQuery. In *Companion Proceedings of the Web Conference 2022 (WWW)*, pages 478–487. ACM, 2022.
- [Gra14] Carrie Gracie. Hong Kong stages huge National Day democracy protests, 2014. URL: https://web.archive.org/web/20220622124010if_/https://www.bbc.com/news/world-asia-china-29430229 (archived on 2022-06-22).
- [GWZ⁺19] Keke Gai, Yulu Wu, Liehuang Zhu, Meikang Qiu, and Meng Shen. Privacy-Preserving Energy Trading Using Consortium Blockchain in Smart Grid. *IEEE Transactions on Industrial Informatics*, 15(6):3548–3558, 2019.
- [HAL⁺18] Danny Yuxing Huang, Maxwell Matthaios Aliapoulios, Vector Guo Li, Luca Invernizzi, Elie Bursztein, Kylie McRoberts, Jonathan Levin, Kirill Levchenko, Alex C. Snoeren, and Damon McCoy. Tracking Ransomware End-to-end. In *Symposium on Security and Privacy (S&P)*, pages 618–631. IEEE, 2018.

- [Her15] Herald Union. Copyright infringement by illegal file sharing in Germany, 2015. URL: https://web.archive.org/web/20191122224713if_/https://www.herald-union.com/copyright-infringement-by-illegal-file-sharing-in-germany/ (archived on 2019-11-22).
- [HHAZ14] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. FLOW-GUARD: Building Robust Firewalls for Software-Defined Networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 97–102. ACM, 2014.
- [HHS⁺16] Martin Henze, Jens Hiller, Sascha Schmerling, Jan Henrik Ziegeldorf, and Klaus Wehrle. CPPL: Compact Privacy Policy Language. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society (WPES)*, pages 99–110. ACM, 2016.
- [HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse Attacks on Bitcoin’s Peer-to-Peer Network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144. USENIX Association, 2015.
- [HMOV04] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.
- [Hou16] Freedom House. Turkey – Freedom of the Press 2016, 2016. URL: <https://web.archive.org/web/20191122231106/https://freedomhouse.org/report/freedom-press/2016/turkey> (archived on 2019-11-22).
- [HSS01] Uri Hanani, Bracha Shapira, and Peretz Shoval. Information Filtering: Overview of Issues, Research and Systems. *User Modeling and User-Adapted Interaction*, 11(3):203–259, 2001.
- [Hug13] HugPuddle. Apertus – Archive data on your favorite blockchains, 2013. URL: https://web.archive.org/web/20220512100657id_/http://apertus.io (archived on 2022-05-12).
- [HZM⁺19] Ke Huang, Xiaosong Zhang, Yi Mu, Xiaofen Wang, Guomin Yang, Xiaojiang Du, Fatemeh Rezaeibagha, Qi Xia, and Mohsen Guizani. Building Redactable Consortium Blockchain for Industrial Internet-of-Things. *Transactions on Industrial Informatics*, 15(6):3670–3679, 2019.
- [HZM⁺20] Ke Huang, Xiaosong Zhang, Yi Mu, Fatemeh Rezaeibagha, Xiaojiang Du, and Nadra Guizani. Achieving Intelligent Trust-Layer for Internet-of-Things via Self-Redactable Blockchain. *Transactions on Industrial Informatics*, 16(4):2677–2686, 2020.
- [IKBS00] Sotiris Ioannidis, Angelos D. Keromytis, Steve M. Bellovin, and Jonathan M. Smith. Implementing a Distributed Firewall. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*, pages 190–199. ACM, 2000.
- [Int15a] Internet Watch Foundation. Hash List, 2015. URL: https://web.archive.org/web/20220313190554id_/https://www.iwf.org.uk/our-technology/our-services/image-hash-list (archived on 2022-03-13).

- [Int15b] INTERPOL. INTERPOL cyber research identifies malware threat to virtual currencies, 2015. URL: https://web.archive.org/web/20220505100920if_/https://www.interpol.int/News-and-Events/News/2015/INTERPOL-cyber-research-identifies-malware-threat-to-virtual-currencies (archived on 2022-05-05).
- [ISB98] Child Trafficking and Pornography Act, 1998. URL: <http://www.irishstatutebook.ie/eli/1998/act/22/enacted/en/pdf>.
- [JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, 2001.
- [JSK19] Hussam Juma, Khaled Shaalan, and Ibrahim Kamel. A Survey on Using Blockchain in Trade Supply Chain Solutions. *IEEE Access*, 7:184115–184132, 2019.
- [JSZ⁺21] Yanxue Jia, Shi-Feng Sun, Yi Zhang, Zhiqiang Liu, and Dawu Gu. Redactable Blockchain Supporting Supervision and Self-Management. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIACCS)*, pages 844–858. ACM, 2021.
- [JZS⁺17] Aljosha Judmayer, Alexei Zamyatin, Nicholas Stifter, Artemios G. Voyiatzis, and Edgar Weippl. Merged Mining: Curse or Cure? In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Hannes Hartenstein, and Jordi Herrera-Joancomartí, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 316–333. Springer Cham, 2017.
- [KAKC20] John Kolb, Moustafa AbdelBaky, Randy H. Katz, and David E. Culler. Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial. *ACM Comput. Surv.*, 53(1), February 2020.
- [Kal17] Benedikt Kalde. Efficient and Secure Compaction of Prunable Blockchains. Master’s thesis, RWTH Aachen University, September 2017.
- [Kes22] Gary C. Kessler. GCK’s File Signatures Table – 19 August 2022, 2022. URL: https://web.archive.org/web/20220930144531if_/https://www.garykessler.net/library/file_sigs.html (archived on 2022-09-30).
- [KJG⁺18] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *Symposium on Security and Privacy (S&P)*, pages 583–598. IEEE, 2018.
- [KK15] Christian Karam and Vitaly Kamluk. Black Hat Asia 2015 Briefings: Decentralized Malware on the Blockchain, 2015. URL: https://web.archive.org/web/20220505100641id_/https://www.blackhat.com/asia-15/briefings.html#decentralized-malware-on-the-blockchain (archived on 2022-05-05).
- [KKSK19] Yujin Kwon, Hyounghick Kim, Jinwoo Shin, and Yongdae Kim. Bitcoin vs. Bitcoin Cash: Coexistence or Downfall of Bitcoin Cash? In *Symposium on Security and Privacy (S&P)*, pages 935–951. IEEE, 2019.

- [KKZ20] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof-of-Burn. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security (FC)*, pages 523–540. Springer Cham, 2020.
- [KMZ20] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive Proofs of Proof-of-Work. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security (FC)*, pages 505–522. Springer Cham, 2020.
- [Kob87] Neal Koblitz. Elliptic Curve Cryptosystems. *Math. Comp.*, 48:203–209, 1987.
- [KPCV14] Dániel Kondor, Márton Pósfai, István Csabai, and Gábor Vattay. Do the Rich Get Richer? An Empirical Analysis of the Bitcoin Transaction Network. *PLOS ONE*, 9(2):1–10, 02 2014.
- [KR00] Hugo Krawczyk and Tal Rabin. Chameleon Signatures. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2000.
- [KWSW07] Mirko Knoll, Arno Wacker, Gregor Schiele, and Torben Weis. Decentralized Bootstrapping in Pervasive Applications. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW)*, pages 589–592. IEEE, 2007.
- [LB15] Kate Lyons and Garry Blight. Where in the world is the worst place to be a Christian?, 2015. URL: https://web.archive.org/web/20220516120111if_/https://www.theguardian.com/world/ng-interactive/2015/jul/27/where-in-the-world-is-it-worst-place-to-be-a-christian (archived on 2022-05-16).
- [LC15a] Antoine Le Calvez. bitcoin-blockchain-parser, 2015. URL: https://web.archive.org/web/20220726204028if_/https://github.com/alectalve/python-bitcoin-blockchain-parser (archived on 2022-07-26).
- [LC15b] Antoine Le Calvez. Non-standard P2SH scripts, 2015. URL: https://web.archive.org/web/20220512094322id_/https://medium.com/@alcio/non-standard-p2sh-scripts-508fa6292df5 (archived on 2022-05-12).
- [LCE18] Tancrede Lepoint, Gabriela Ciocarlie, and Karim Eldefrawy. BlockCIS—A Blockchain-Based Cyber Insurance System. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 378–384. IEEE, 2018.
- [Lee13] David Lee. France ends three-strikes internet piracy ban policy, 2013. URL: https://web.archive.org/web/20220514115330if_/https://www.bbc.com/news/technology-23252515 (archived on 2022-05-14).
- [LHAG19] Hongfang Lu, Kun Huang, Mohammadamin Azimi, and Lijun Guo. Blockchain Technology in the Oil and Gas Industry: A Review of Applications, Opportunities, Challenges, and Risks. *IEEE Access*, 7:41426–41444, 2019.
- [LKBM17] Isis Lovecruft, George Kadianakis, Ola Bini, and Nick Mathewson. Tor Guard Specification, 2017. URL: https://web.archive.org/web/20230307143702id_/https://gitweb.torproject.org/torspec.git/plain/guard-spec.txt (archived on 2023-03-07).

- [LLW15] Eric Lombrozo, Johnson Lau, and Pieter Wuille. BIP 141: Segregated Witness (Consensus layer), 2015. URL: https://web.archive.org/web/20220110130012if_/https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (archived on 2022-01-10).
- [LNZ⁺16] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 17–30. ACM, 2016.
- [Lop20] Jameson Lopp. 2020 Bitcoin Node Performance Tests, 2020. URL: <https://web.archive.org/web/20210202105912/https://blog.lopp.net/2020-bitcoin-node-performance-tests> (archived on 2021-02-02).
- [LPMW22] Johannes Lohmöller, Jan Pennekamp, Roman Matzutt, and Klaus Wehrle. On the Need for Strong Sovereignty in Data Ecosystems. In *Proceedings of the First International Workshop on Data Ecosystems (DEco)*. VLDB Emdowment, 2022.
- [LPP19] Taeho Lee, Christos Pappas, and Adrian Perrig. Bootstrapping Privacy Services in Today’s Internet. *SIGCOMM Comput. Commun. Rev.*, 48(5):21–30, 1 2019.
- [LSGZ19] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. Vault: Fast Bootstrapping for the Algorand Cryptocurrency. In *26th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2019.
- [LSM06] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A Survey of Solutions to the Sybil Attack. Technical report, University of Massachusetts Amherst, 2006, URL: <http://forensics.umass.edu/pubs/levine.sybil.tr.2006.pdf>.
- [Luo22] Bin Luo. Efficient and Fine-grained Redactable Blockchain Supporting Accountability and Updating Policies, 2022. URL: <https://arxiv.org/abs/2211.08813>. arXiv:2211.08813.
- [LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting Cryptographic Key Sizes. *J. Cryptology*, 14(4):255–293, 2001.
- [LW16] Eric Lombrozo and Pieter Wuille. BIP 144: Segregated Witness (Peer Services), 2016. URL: http://web.archive.org/web/20211225091505if_/https://github.com/bitcoin/bips/blob/master/bip-0144.mediawiki (archived on 2021-12-25).
- [LXY⁺22] Xin-Yu Li, Jing Xu, Ling-Yuan Yin, Yuan Lu, Qiang Tang, and Zhen-Feng Zhang. Escaping From Consensus: Instantly Redactable Blockchain Protocols in Permissionless Setting. *Transactions on Dependable and Secure Computing*, 2022. Early Access Version.
- [Lyn13] Lisa Lynch. *The Leak Heard Round the World? Cablegate in the Evolving Global Mediascape*, pages 56–77. Palgrave Macmillan UK, 2013.
- [LYOK19] Nam-Yong Lee, Jinhong Yang, Md Mehedi Hassan Onik, and Chul-Soo Kim. Modifiable Public Blockchains Using Truncated Hashing and Sidechains. *IEEE Access*, 7:173571–173582, 2019.

- [LZS⁺18] Hongyu Li, Liehuang Zhu, Meng Shen, Feng Gao, Xiaoling Tao, and Sheng Liu. Blockchain-Based Data Preservation System for Medical Data. *Journal of Medical Systems*, 42(8):141, 2018.
- [MAP⁺22a] Roman Matzutt, Vincent Ahlrichs, Jan Pennekamp, Roman Karwacik, and Klaus Wehrle. A Moderation Framework for the Swift and Transparent Removal of Illicit Blockchain Content. In *Proceedings of the International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2022.
- [MAP⁺22b] Roman Matzutt, Vincent Ahlrichs, Jan Pennekamp, Roman Karwacik, and Klaus Wehrle. RedactChain: Proof-of-Concept Prototype for the Swift and Transparent Removal of Illicit Blockchain Content, March 2022. URL: <https://github.com/COMSYS/redactchain>.
- [Mat14] Jon Matonis. The Bitcoin Mining Arms Race: GHash.io and the 51% Issue, 2014. URL: https://web.archive.org/web/20220411094716if_/https://www.coindesk.com/markets/2014/07/17/the-bitcoin-mining-arms-race-ghashio-and-the-51-issue (archived on 2022-04-11).
- [Max13a] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world, 2013. URL: https://web.archive.org/web/20220201100049if_/https://bitcointalk.org/index.php?topic=279249 (archived on 2021-02-01).
- [Max13b] Gregory Maxwell. To prevent arbitrary data storage in txouts – The Ultimate Solution, 2013. URL: https://web.archive.org/web/20210308031608if_/https://sourceforge.net/p/bitcoin/mailman/message/30705609 (archived on 2021-03-08).
- [MB09] Prateek Mittal and Nikita Borisov. ShadowWalker: Peer-to-Peer Anonymous Communication Using Redundant Structured Topologies. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 161–172. ACM, 2009.
- [MB15] Malte Möser and Rainer Böhme. Trends, Tips, Tolls: A Longitudinal Study of Bitcoin Transaction Fees. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security (FC)*, pages 19–33. Springer Berlin Heidelberg, 2015.
- [MBB13] Malte Möser, Rainer Böhme, and Dominic Breuker. An Inquiry into Money Laundering Tools in the Bitcoin Ecosystem. In *APWG eCrime Researchers Summit*, 2013.
- [MCR⁺20] Sami Ben Mariem, Pedro Casas, Matteo Romiti, Benoit Donnet, Rainer Stütz, and Bernhard Haslhofer. All that Glitters is not Bitcoin – Unveiling the Centralized Nature of the BTC (IP) Network. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2020.
- [Mer87] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 369–378. Springer Berlin Heidelberg, 1987.
- [MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *Symposium on Security and Privacy (S&P)*, pages 397–411. IEEE, 2013.

- [MHH⁺16] Roman Matzutt, Oliver Hohlfeld, Martin Henze, Robin Rawiel, Jan Henrik Ziegeldorf, and Klaus Wehrle. POSTER: I Don't Want That Content! On the Risks of Exploiting Bitcoin's Blockchain as a Content Store. In *Proceedings of the Conference on Computer and Communications Security (CCS)*, pages 1769–1771. ACM, 2016.
- [MHH⁺18] Roman Matzutt, Jens Hiller, Martin Henze, Jan Henrik Ziegeldorf, Dirk Müllmann, Oliver Hohlfeld, and Klaus Wehrle. A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)*, pages 420–438. Springer Berlin Heidelberg, 2018.
- [MHMW24] Roman Matzutt, Martin Henze, Dirk Müllmann, and Klaus Wehrle. Illicit Blockchain Content: Its Different Shapes, Consequences, and Remedies. In Sushmita Ruj, Salil Kanhere, and Mauro Conti, editors, *Blockchains – A Handbook on Fundamentals, Platforms, and Applications*, chapter 10, pages 301–306. Springer Cham, 2024.
- [MHZ⁺18] Roman Matzutt, Martin Henze, Jan Henrik Ziegeldorf, Jens Hiller, and Klaus Wehrle. Thwarting Unwanted Blockchain Content Insertion. In *Proceedings of the International Conference on Cloud Engineering (IC2E)*, pages 364–370. IEEE, 2018.
- [Mil85] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426. Springer Berlin Heidelberg, 1985.
- [Mit94] Nilotpala Mitra. Efficient Encoding Rules for ASN.1-based Protocols. *AT&T Technical Journal*, 73(3):80–93, 1994.
- [MJP⁺20] Michael Mirkin, Yan Ji, Jonathan Pang, Arian Klages-Mundt, Ittay Eyal, and Ari Juels. BDoS: Blockchain Denial-of-Service. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 601–619. ACM, 2020.
- [MKKS15] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourcable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 680–691. ACM, 2015.
- [MKP⁺20a] Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle. How to Securely Prune Bitcoin's Blockchain. In *Proceedings of the Networking Conference (Networking)*, pages 298–306. IFIP, 2020.
- [MKP⁺20b] Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle. CoinPrune - Deployable Block Pruning for Bitcoin, April 2020. URL: <https://github.com/COMSYS/coinprune>.
- [MKP⁺21] Roman Matzutt, Benedikt Kalde, Jan Pennekamp, Arthur Drichel, Martin Henze, and Klaus Wehrle. CoinPrune: Shrinking Bitcoin's Blockchain Retrospectively. *IEEE Transactions on Network and Service Management*, 18(3):3064–3078, 2021.

- [MLS⁺15] Emily McReynolds, Adam Lerner, Will Scott, Franziska Roesner, and Tadayoshi Kohno. Cryptographic Currencies from a Tech-Policy Perspective: Policy Issues and Technical Directions. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security (FC)*, pages 94–111. Springer Berlin Heidelberg, 2015.
- [MM18] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless Tumbling for Transaction Privacy. *Proc. Priv. Enhancing Technol.*, 2018(2):105–121, 2018.
- [MN22] Malte Möser and Arvind Narayanan. Resurrecting Address Clustering in Bitcoin. In Ittay Eyal and Juan Garay, editors, *Financial Cryptography and Data Security (FC)*, pages 386–403. Springer Cham, 2022.
- [MNB⁺18] Esther Mengelkamp, Benedikt Notheisen, Carolin Beer, David Dauer, and Christof Weinhardt. A blockchain-based smart grid: towards sustainable local energy markets. *Computer Science - Research and Development*, 33(1):207–214, 2018.
- [MNLX22] Xianning Meng, Peifang Ni, Hongda Li, and Haixian Xu. A Conflict-Free Redactable Blockchain. In *International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 285–290. IEEE, 2022.
- [Mon19a] Money Button. Against Illegal Content on the Blockchain, 2019. URL: https://web.archive.org/web/20210128090213if_/https://blog.moneybutton.com/2019/01/31/against-illegal-content-on-the-blockchain/ (archived on 2021-01-28).
- [Mon19b] Money Button. How We Added Support for Giant OP_RETURN Data in Money Button, 2019. URL: https://web.archive.org/web/20210121180521if_/https://blog.moneybutton.com/2019/01/26/how-we-added-support-for-giant-op_return-data-in-money-button/ (archived on 2021-01-21).
- [Mor17] Daniel Morgan. The Great Bitcoin Scaling Debate – A Timeline, 2017. URL: https://web.archive.org/web/20210326171612id_/https://hackernoon.com/the-great-bitcoin-scaling-debate-a-timeline-6108081dbada (archived on Accessed 2021-03-26).
- [MPBW20a] Roman Matzutt, Jan Pennekamp, Erik Buchholz, and Klaus Wehrle. Utilizing Public Blockchains for the Sybil-Resistant Bootstrapping of Distributed Anonymity Services. In *Proceedings of the 15th Asia Conference on Computer and Communications Security (ASIACCS)*, pages 531–542. ACM, 2020.
- [MPBW20b] Roman Matzutt, Jan Pennekamp, Erik Buchholz, and Klaus Wehrle. AnonBoot - Utilizing Public Blockchains for the Sybil-Resistant Bootstrapping of Distributed Anonymity Services, April 2020. URL: <https://github.com/COMSYS/anonboot>.
- [MPJ⁺13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. A Fistful of Bitcoins: Characterizing Payments among Men with No Names. In *Proceedings of the*

- 2013 *Conference on Internet Measurement Conference (IMC)*, pages 127–140. ACM, 2013.
- [MPW20] Roman Matzutt, Jan Pennekamp, and Klaus Wehrle. A Secure and Practical Decentralized Ecosystem for Shareable Education Material. In *Proceedings of the 34th International Conference on Information Networking (ICOIN)*, pages 529–534. IEEE, 2020.
- [MSTP⁺22] Garrett Morrow, Briony Swire-Thompson, Jessica Montgomery Polny, Matthew Kopec, and John P. Wihbey. The emerging science of content labeling: Contextualizing social media content moderation. *Journal of the Association for Information Science and Technology*, 73(10):1365–1386, 2022.
- [MWA⁺18] Jan Mendling, Ingo Weber, Wil van der Aalst, Jan vom Brocke, Cristina Cabanillas, Florian Daniel, Søren Debois, Di Claudio Ciccio, Marlon Dumas, Schahram Dustdar, Avigdor Gal, Luciano García-Bañuelos, Guido Governatori, Richard Hull, Marcello La Rosa, Henrik Leopold, Frank Leymann, Jan Recker, Manfred Reichert, Hajo A. Reijers, Stefanie Rinderle-Ma, Andreas Solti, Michael Rosemann, Stefan Schulte, Munindar P. Singh, Tijs Slaats, Mark Staples, Barbara Weber, Matthias Weidlich, Mathias Weske, Xiwei Xu, and Liming Zhu. Blockchains for Business Process Management - Challenges and Opportunities. *ACM Transactions on Management Information Systems*, 9(1), February 2018.
- [MZ19] Alexander Marsalek and Thomas Zefferer. A Correctable Public Blockchain. In *18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*, pages 554–561. IEEE, 2019.
- [MZFZ19] Alexander Marsalek, Thomas Zefferer, Edona Fasllija, and Dominik Ziegler. Tackling Data Inefficiency: Compressing the Bitcoin Blockchain. In *18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*, pages 626–633. IEEE, 2019.
- [MZN⁺21] Muhammad Baqer Mollah, Jun Zhao, Dusit Niyato, Kwok-Yan Lam, Xin Zhang, Amer M. Y. M. Ghias, Leong Hai Koh, and Lei Yang. Blockchain for Future Smart Grid: A Comprehensive Survey. *IEEE Internet of Things Journal*, 8(1):18–43, 2021.
- [Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. White paper, 2008, URL: <https://bitcoin.org/bitcoin.pdf>.
- [Nak10] Satoshi Nakamoto. Bitcoin 0.3.2 released, 2010. URL: https://web.archive.org/web/20230107153157if_/https://bitcointalk.org/index.php?topic=437 (archived on 2023-01-07).
- [NR95] Kaisa Nyberg and Rainer A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT’94*, pages 182–193. Springer Berlin Heidelberg, 1995.

- [NS21] Satoshi Nakamoto and Manu Sporny. The Base58 Encoding Scheme (draft-msporny-base58-03), 2021. URL: https://web.archive.org/web/20220110130326id_/https://tools.ietf.org/id/draft-msporny-base58-03.txt (archived on 2022-01-10). IETF Internet Draft.
- [O’B19] James O’Beirne. Assume UTXO, 2019. URL: <https://web.archive.org/web/20230107160709/https://github.com/jamesob/assumeutxo-docs/tree/master/proposal> (archived on 2023-01-07).
- [OKH13] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and Anonymity of the Bitcoin Transaction Graph. *Future Internet*, 5(2):237–250, 2013.
- [PAM⁺20] Jan Pennekamp, Fritz Alder, Roman Matzutt, Jan Tobias Mühlberg, Frank Piessens, and Klaus Wehrle. Secure End-to-End Sensing in Supply Chains. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–6. IEEE, 2020.
- [Par18] Juha Partala. Provably Secure Covert Communication on Blockchain. *Cryptography*, 2(3), 2018.
- [PCM11] Siani Pearson and Marco Casassa-Mont. Sticky Policies: An Approach for Managing Privacy across Multiple Parties. *Computer*, 44(9):60–68, 2011.
- [PDC17] Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. μ chain: How to Forget without Hard Forks. Cryptology ePrint Archive, Report 2017/106, 2017. URL: <https://ia.cr/2017/106>. Version 20200602:075626.
- [Ped91] Torben Pryds Pedersen. A Threshold Cryptosystem without a Trusted Party. In Donald W. Davies, editor, *Advances in Cryptology (EUROCRYPT)*, pages 522–526. Springer Berlin Heidelberg, 1991.
- [Pee05] Randall Peerenboom. Assessing Human Rights in China: Why the Double Standard. *Cornell International Law Journal*, 38(1):71–172, 2005.
- [PHS18] Moritz Petersen, Niels Hackius, and Birgit von See. Mapping the sea of opportunities: Blockchain in supply chain and logistics. *it - Information Technology*, 60(5-6):263–271, 2018.
- [PJJ17] Sungmi Park, Yunsik Jake Jang, and Joshua I. James. Possession of Child Exploitation Material in Computer Temporary Internet Cache. *Journal of Digital Forensics, Security and Law*, 12:7–22, 09 2017.
- [PK01] Andreas Pfitzmann and Marit Köhntopp. Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, pages 1–9. Springer Berlin Heidelberg, 2001.
- [Poe16] Andrew Poelstra. Mumblewimble. White paper, 2016, URL: <https://download.wpsoftware.net/bitcoin/wizardry/mumblewimble.pdf>.
- [Pri20] Privacy Pros. Bitcoin Transaction Fee Estimator & Calculator, 2020. URL: https://web.archive.org/web/20221006150145if_/https://privacypros.io/tools/bitcoin-fee-estimator/ (archived on 2022-10-06).

- [PRR09] Andriy Panchenko, Stefan Richter, and Arne Rache. NISAN: Network Information Service for Anonymization Networks. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 141–150. ACM, 2009.
- [PSB18] Emanuel Palm, Olov Schelén, and Ulf Bodin. Selective Blockchain Transaction Pruning and State Derivability. In *Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 31–40. IEEE, 2018.
- [Raw16] Robin Rawiel. Analysis and Classification of Arbitrary Data Storage in Bitcoin’s Blockchain. Bachelor’s thesis, RWTH Aachen University, October 2016.
- [Rei12] Alan Reiner. Ultimate blockchain compression w/ trust-free lite nodes, 2012. URL: https://web.archive.org/web/20230107145658if_/https://bitcointalk.org/index.php?topic=88208 (archived on 2023-01-07).
- [RH13] Fergal Reid and Martin Harrigan. An Analysis of Anonymity in the Bitcoin System. In Yaniv Altshuler, Yuval Elovici, Armin B. Cremers, Nadav Aharony, and Alex Pentland, editors, *Security and Privacy in Social Networks*, chapter 10, pages 197–223. Springer New York, 2013.
- [RKY+20] Scott Ruoti, Ben Kaiser, Arkady Yerukhimovich, Jeremy Clark, and Robert Cunningham. Blockchain Technology: What is It Good For? *Commun. ACM*, 63(1):46–53, 1 2020.
- [RMSK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In Mirosław Kutylowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 345–364. Springer Cham, 2014.
- [Rod21] Przemysław Rodwald. An Analysis of Data Hidden in Bitcoin Addresses. In Wojciech Zamojski, Jacek Mazurkiewicz, Jarosław Sugier, Tomasz Walkowiak, and Janusz Kacprzyk, editors, *Theory and Engineering of Dependable Computer Systems and Networks*, pages 369–379. Springer Cham, 2021.
- [Roe99] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration (LISA)*, pages 229–238. USENIX Association, 1999.
- [RS13] Dorit Ron and Adi Shamir. Quantitative Analysis of the Full Bitcoin Transaction Graph. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography and Data Security (FC)*, pages 6–24. Springer Berlin Heidelberg, 2013.
- [RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.
- [SA21] Andrew Spurr and Marcel Ausloos. Challenging practical features of Bitcoin by the main altcoins. *Quality & Quantity*, 55(5):1541–1559, 2021.
- [Sab13] Nicolas van Saberhagen. CryptoNote v 2.0. White paper, 2013, URL: https://web.archive.org/web/20201028121818id_/https://cryptonote.org/whitepaper.pdf (archived on 2020-10-28).

- [SBHD17] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquenoy. Towards Blockchain-Based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 on Cloud Computing Security Workshop (CCSW)*, pages 45–50. ACM, 2017.
- [Sch15] Samantha H. Scheller. A Picture Is Worth a Thousand Words: The Legal Implications of Revenge Porn. *North Carolina Law Review*, 93(2):551–595, 2015.
- [Sch19] Bruce Schneier. There’s No Good Reason to Trust Blockchain Technology, 2019. URL: https://web.archive.org/web/20211028090733if_/https://www.wired.com/story/theres-no-good-reason-to-trust-blockchain-technology (archived on 2021-10-28).
- [SGGZ19] Andrew Stone, Joshua Green, Andrew Groot, and Christopher Zeng. Bitcoin Cash Protocol: Network-Level Validation Rules, 2019. URL: https://web.archive.org/web/20220523091441if_/https://reference.cash/protocol/blockchain/transaction-validation/network-level-validation-rules (archived on 2022-05-23).
- [Sha79] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, nov 1979.
- [Sha18] Hamza Shaban. People are using bitcoin’s system to share child pornography, researchers say, 2018. URL: https://web.archive.org/web/20220505111009if_/https://www.washingtonpost.com/news/the-switch/wp/2018/03/22/people-are-using-bitcoins-system-to-share-child-pornography/?variant=95d42e19c24b03e7 (archived on 2022-05-05).
- [She19] Andrey Shevchenko. Monero Penalizes GPU and ASIC Mining with RandomX Upgrade, 2019. URL: https://web.archive.org/web/20230307140633if_/https://cryptobriefing.com/monero-penalizes-gpu-mining-randomx (archived on 2023-03-07).
- [Shi14] Ken Shirriff. Hidden surprises in the Bitcoin blockchain and how they are stored: Nelson Mandela, Wikileaks, photos, and Python software, 2014. URL: https://web.archive.org/web/20210625135739if_/http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photographs.html (archived on 2021-06-25).
- [SIO20] Teppei Sato, Mitsuyoshi Imamura, and Kazumasa Omote. Threat Analysis of Poisoning Attack Against Ethereum Blockchain. In Maryline Laurent and Thanassis Giannetsos, editors, *Information Security Theory and Practice*, pages 139–154. Springer Cham, 2020.
- [SLY15] Matthew D. Sleiman, Adrian P. Lauf, and Roman Yampolskiy. Bitcoin Message: Data Insertion on a Proof-of-Work Cryptocurrency System. In *2015 International Conference on Cyberworlds (CW)*, pages 332–336. IEEE, 2015.
- [SM19] Herman Schoenfeld and Albert Molina. Pascal: An Infinitely Scalable Cryptocurrency. White paper, 2019, URL: <https://www.pascalcoin.org/storage/whitepapers/PascalWhitePaperV5.pdf>.

- [SMZ14] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. BitIodine: Extracting Intelligence from the Bitcoin Network. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security (FC)*, pages 457–468. Springer Berlin Heidelberg, 2014.
- [Son19] Jimmy Song. *Programming Bitcoin*. O’Reilly, first edition, 2019.
- [SSZ17] Ayelet Sapirshstein, Yonatan Sompolinsky, and Aviv Zohar. Optimal Selfish Mining Strategies in Bitcoin. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security (FC)*, pages 515–532. Springer Berlin Heidelberg, 2017.
- [StGB] German Criminal Code (Strafgesetzbuch), English Translation, 2019. URL: https://web.archive.org/web/20220518093201if_/https://www.gesetze-im-internet.de/englisch_stgb/englisch_stgb.html (archived on 2022-05-18). Criminal Code in the version published on 13 November 1998 (Federal Law Gazette I, p. 3322), as last amended by Article 2 of the Act of 19 June 2019 (Federal Law Gazette I, p. 844). Translated by Michael Bohlander and Ute Reusch.
- [SUP04] Ali Aydın Selçuk, Ersin Uzun, and Mark Reşat Pariente. A Reputation-Based Trust Management System for P2P Networks. In *International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 251–258. IEEE, 2004.
- [SVS18] Andrew Sward, Ivy Vecna, and Forrest Stonedahl. Data Insertion in Bitcoin’s Blockchain. *Ledger*, 3, Apr. 2018.
- [Sza94] Nick Szabo. Smart Contracts, 1994. URL: https://web.archive.org/web/20160323035617id_/http://szabo.best.vwh.net/smart.contracts.html (archived on 2016-03-23).
- [Szi15] Péter Szilágyi. eth/63 fast synchronization algorithm, 2015. URL: https://web.archive.org/web/20230107154105if_/https://github.com/ethereum/go-ethereum/pull/1889 (archived on 2023-01-07).
- [TBM⁺21] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Bernardo Magri, Daniel Tschudi, and Aniket Kate. Reparo: Publicly verifiable layer to repair blockchains. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security (FC)*, pages 37–56. Springer Berlin Heidelberg, 2021.
- [TD17] Alin Tomescu and Srinivas Devadas. Catena: Efficient Non-equivocation via Bitcoin. In *Symposium on Security and Privacy (S&P)*, pages 393–409. IEEE, 2017.
- [Tel21] Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T). X.690: Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). Standard, feb 2021, URL: <https://handle.itu.int/11.1002/1000/14472>.
- [Tev18] tevador. RandomX, 2018. URL: https://web.archive.org/web/20230307141740if_/https://github.com/tevador/RandomX (archived on 2023-03-07).

- [The17] Alexander Theißen. Decentralized Bootstrapping for SMC-based Services. Master’s thesis, RWTH Aachen University, May 2017.
- [THK⁺18] Muhamed Turkanović, Marko Hölbl, Kristjan Košič, Marjan Heričko, and Aida Kamišalić. EduCTX: A Blockchain-Based Higher Education Credit Platform. *IEEE Access*, 6:5112–5127, 2018.
- [TLL⁺20] Yangguang Tian, Nan Li, Yingjiu Li, Pawel Szalachowski, and Jianying Zhou. Policy-Based Chameleon Hash for Blockchain Rewriting with Black-Box Accountability. In *Annual Computer Security Applications Conference (ACSAC)*, pages 813–828. ACM, 2020.
- [Tod12] Peter Todd. Merkle Mountain Ranges, 2012. URL: <https://web.archive.org/web/20230107210502/https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md> (archived on 2023-01-07).
- [Tor06] Roger Dingledine and Nick Mathewson. Tor Path Specification, 2006. URL: https://web.archive.org/web/20230306085007id_/https://gitweb.torproject.org/torspec.git/plain/path-spec.txt (archived on 2023-03-06).
- [Tor07] The Tor Project. Tor directory protocol, version 3, 2007. URL: https://web.archive.org/web/20230302120507id_/https://gitweb.torproject.org/torspec.git/plain/dir-spec.txt (archived on 2023-03-02).
- [Tor09] The Tor Project. Tor Metrics – Relay Search (flag: Authority), 2009. URL: <https://metrics.torproject.org/rs.html#search/flag:Authority>.
- [Tor14] The Tor Project. Reporting Bad Relays, 2014. URL: https://web.archive.org/web/20230221123833id_/https://gitlab.torproject.org/legacy/trac/-/wikis/doc/ReportingBadRelays (archived on 2023-02-21).
- [Tro15] John Tromp. Cuckoo Cycle: A Memory Bound Graph-Theoretic Proof-of-Work. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security (FC)*, pages 49–62. Springer Berlin Heidelberg, 2015.
- [TS11] Amir Taaki and Patrick Strateman. BIP 14: Protocol Version and User Agent, 2011. URL: https://web.archive.org/web/20220325020723if_/https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki (archived on 2022-03-25).
- [TS16] Florian Tschorsch and Björn Scheuermann. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys & Tutorials*, 18(3):2084–2123, 2016.
- [Tuc16] Eric Tucker. A Look at Federal Cases on Handling Classified Information, 2016. URL: https://web.archive.org/web/20210216144814if_/http://www.military.com/daily-news/2016/01/30/a-look-at-federal-cases-on-handling-classified-information.html (archived on 2021-02-16).
- [UKPGA78] Protection of Children Act 1978, 2022. URL: https://web.archive.org/web/20220519114401if_/https://www.legislation.gov.uk/ukpga/1978/37 (archived on 2022-05-19).

- [Un00a] United Nations. Optional Protocol to the Convention on the Rights of the Child on the sale of children, child prostitution and child pornography, 2000. URL: https://treaties.un.org/doc/Treaties/2000/05/20000525%2003-16%20AM/Ch_IV_11_cp.pdf. The Optional Protocol was adopted by resolution A/RES/54/263 of 25 May 2000 at the fifty-fourth session of the General Assembly of the United Nation.
- [Un00b] United Nations. Status of Treaties: Optional Protocol to the Convention on the Rights of the Child on the sale of children, child prostitution and child pornography, 2000. URL: <https://treaties.un.org/doc/Publication/MTDSG/Volume%20I/Chapter%20IV/IV-11-c.en.pdf>.
- [Unw19] unwriter. B:// – Bitcoin Data Protocol, 2019. URL: https://web.archive.org/web/20220512112519if_/https://github.com/unwriter/B (archived on 2022-05-12).
- [U.S.C.] United States Code, 2022. URL: https://web.archive.org/web/20220519113315id_/http://uscode.house.gov/view.xhtml?req=granuleid%3AUSC-prelim-title18-section2256&num=0&edition=prelim (archived on 2022-05-19).
- [VCSA17] Jessica Vitak, Kalyani Chadha, Linda Steiner, and Zahra Ashktorab. Identifying Women’s Experiences With and Strategies for Mitigating Negative Effects of Online Harassment. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*, pages 1231–1245. ACM, 2017.
- [VM15] Marie Vasek and Tyler Moore. There’s No Free Lunch, Even Using Bitcoin: Tracking the Popularity and Profits of Virtual Currency Scams. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security (FC)*, pages 44–61. Springer Berlin Heidelberg, 2015.
- [VXBS20] Wattana Viriyasitavat, Li Da Xu, Zhuming Bi, and Assadaporn Sapsomboon. Blockchain-based business process management (BPM) framework for service composition in industry 4.0. *Journal of Intelligent Manufacturing*, 31(7):1737–1748, 2020.
- [WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction of algebraic blockcodes, 1986. US Patent 4,633,470.
- [Wei14] Wang Wei. Ancient ‘STONED’ Virus Signatures found in Bitcoin Blockchain, 2014. URL: https://web.archive.org/web/20220513134518if_/https://thehackernews.com/2014/05/microsoft-security-essential-found.html (archived on 2022-05-13).
- [WEWH22] Mark Webber, Vincent Elfving, Sebastian Weidt, and Winfried K. Hensinger. The impact of hardware specifications on reaching quantum advantage in the fault tolerant regime. *AVS Quantum Science*, 4(1), 2022.
- [WG18] Karl Wüst and Arthur Gervais. Do you Need a Blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

- [WKM⁺14] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled Onions: Exposing Malicious Tor Exit Relays. In Emiliano De Cristofaro and Steven J. Murdoch, editors, *Privacy Enhancing Technologies*, pages 304–331. Springer Cham, 2014.
- [WMP⁺22] Eric Wagner, Roman Matzutt, Jan Pennekamp, Lennart Bader, Irakli Bajelidze, Klaus Wehrle, and Martin Henze. Scalable and Privacy-Focused Company-Centric Supply Chain Management. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2022.
- [Woo14] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. White paper, 2014, URL: <https://ethereum.github.io/yellowpaper/paper.pdf>.
- [Wor20] World Economic Forum (WEF). Global Standards Mapping Initiative: An overview of blockchain technical standards. Technical report, 2020, URL: http://www3.weforum.org/docs/WEF_GSMI_Technical_Standards_2020.pdf.
- [WRC00] Marc Waldman, Aviel D. Rubin, and Lorrie F. Cranor. Publius: A Robust, Tamper-Evident, Censorship-Resistant, and Source-Anonymous Web Publishing System. In *9th USENIX Security Symposium (USENIX Security)*. USENIX Association, 2000.
- [WS06] Kevin Walsh and Emin Gün Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation (NSDI)*, pages 1–14. USENIX Association, 2006.
- [WTMR15] Pieter Wuille, Peter Todd, Gregory Maxwell, and Rusty Russell. BIP 9: Version bits with timeout and delay, 2015. URL: https://web.archive.org/web/20220110124425if_/https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki (archived on 2022-01-10).
- [XHY⁺23] Shengmin Xu, Xinyi Huang, Jiaming Yuan, Yingjiu Li, and Robert H. Deng. Accountable and Fine-Grained Controllable Rewriting in Blockchains. *Transactions on Information Forensics and Security*, 18:101–116, 2023.
- [YMRS18] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. NISTIR 8202: Blockchain Technology Overview. Nist internal report, 2018, URL: <https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf>.
- [YSJA21] Ibrar Yaqoob, Khaled Salah, Raja Jayaraman, and Yousof Al-Hammadi. Blockchain for healthcare data management: opportunities, challenges, and future recommendations. *Neural Computing and Applications*, 2021.
- [ZGH⁺15] Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, and Klaus Wehrle. CoinParty: Secure Multi-Party Mixing of Bitcoins. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 75–86. ACM, 2015.
- [ZMH⁺18] Jan Henrik Ziegeldorf, Roman Matzutt, Martin Henze, Fred Grossmann, and Klaus Wehrle. Secure and anonymous decentralized Bitcoin mixing. *Future Generation Computer Systems*, 80:448–466, 3 2018.

- [ZMR18] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. RapidChain: Scaling Blockchain via Full Sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 931–948. ACM, 2018.
- [ZRWW16] Torsten Zimmermann, Jan Rüth, Hanno Wirtz, and Klaus Wehrle. Maintaining Integrity and Reputation in Content Offloading. In *12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 25–32. IEEE, 2016.
- [ZSJ⁺19] A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottenbelt. A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *Financial Cryptography and Data Security (FC)*, pages 31–42. Springer Berlin Heidelberg, 2019.
- [ZXD⁺18] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: a survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.
- [ZYH⁺22] Wei Zhang, Jiangshan Yu, Qingqiang He, Nan Zhang, and Nan Guan. TICK: Tiny Client for Blockchains. *Internet of Things Journal*, 9(16):14172–14184, 2022.