Master Thesis

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Hardening Bitcoin against off-topic data inclusion attacks

Andreas Seeg
Department of Computer Science
Chair for Computer Science II (Software Engineering)

Prof. Dr. Alexandra Dmitrienko
Advisor

Submission
21. August 2018

www.uni-wuerzburg.de

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Würzburg, 21. August 2018

.....

(Andreas Seeg)

Abstract

Bitcoin is a novel peer to peer payment network that was started in 2009 and that has since flourished, gaining acceptance all over the world. By contributing their ressources such as network connectivity and computing power, all participants are creating the the Bitcoin network together. Designed as an open source software system, the participation rules for this network are not upheld by humans, but implemented in code, and generally allow anyone to join or leave the network at will.

Since the time of Bitcoin's inception, software errors had to be fixed and network attacks that tried to make parts of the network unavailable have been endured. This thesis provides a solution to the problem of illegal data inclusions within the blockchain, a data structure that cannot easily be changed and that has to be transmitted to participants of Bitcoin. The flaw, i.e. the possibility for any user of Bitcoin to potentially include illegal data and therefore illegalizes the whole blockchain, is not just a software flaw, but can be seen as a structural one if free participation and uncensorable payments are to be provided.

The proposal to solve the problem is layered in two tiers. The first tier prevents effective data inclusion for some standard payment functions within Bitcoin. This allows participants to be certain that these transactions do not contain bad data, allowing them to be included into Bitcoin blocks. It also solves the problem of illegal data permanently occupying valuable ressources on Bitcoin nodes, as parts of the provably clean transactions are generally held in RAM until they have been spent. The prevention mechanism is essentially a system that provides proof that bitcoins were sent to actual Bitcoin addresses, stopping data inclusion by address fields abusal.

The second tier consists of rule changes for Bitcoin. First of all, all non-standard payment transactions are removed from Bitcoin, with the exception of the more computationally flexible and popular P2SH transactions. This limits the avenues for data inclusion in Bitcoin without destroying its flexibility or its smart contract capabilities. Then, P2SH transactions are redesigned to allow for a more fine grained deletion of their data, so only offending parts could be removed without destroying the integrity of the rest. Finally, a special type of transaction is introduced that allows censorship of just the signature data of a P2SH transaction. This transaction has to include a payment equal to the Bitcoin provided by the previous P2SH transaction, however. By adding this requirement, the system is protected from becoming open to money creation attacks that might inflate the currency without the agreement of all participants.

With these changes, illegal data inclusion becomes either impossible, or they can be removed without destorying the integrity of the payment system as a whole or for payments that have already happened.

Implementing all proposed changes would go beyond the scope of a master thesis. Tier 1 of the presented solution has been implemented as a proof-of-concept, though. As a basis, the most popular Bitcoin node software, Bitcoin Core, has been used. Additionally, the ideas behind tier 2 are explained in full, and the potential impact on the system is discussed.

This thesis also covers previous attempts to allow for data removal in similar systems. Their discussion is meant to encourage the adoption of the presented proposal, as it should cause less disruption than some more extreme measurements that might introduce centralization into a system that was meant to run decentralized.

Zusammenfassung

Bitcoin ist ein neuartiges, peer to peer Bezahlnetzwerk das im Jahre 2009 offiziell gestartet wurde. Seitdem hat es sich stark weiterentwickelt und findet in vielen Ländern der Welt verwendung. Das Bitcoin-Netzwerk besteht aus vielen freiwilligen Teilnehmern, die Netzwerk- bzw. Berechnungs-Ressourcen bereitstellen, um es am laufen zu erhalten. Das System wurde als Open Source Software konzipiert, und die Teilnahmeregeln des Netzwerkes werden nicht direkt durch Menschen gesteuert, sondern als Programmcode implementiert, um so jedem praktisch diskriminierungsfrei die Teilnahme zu ermöglichen.

Seit dem Start des Netzwerkes wurden bereits einige Softwarefehler repariert und Netzwerkangriffe, die das Netzwerk unerreichbar machen wollten, wurden ausgehalten oder gestoppt. Diese Abschlussarbeit beschäftigt sich mit dem Problem illegaler Datenablage in der Bitcoin blockchain. Die Blockchain ist eine Datenstruktur, die normalerweise keine nachträglichen Änderungen erlaubt, und die außerdem an die Teilnehmer des Netzwerkes verteilt werden muss. Problematisch ist die Möglichkeit, dass ein belieber Bitcoin-Nutzer potentiell illegale Inhalte in die Blockchain speichert, um so Teile der Blockchain und somit Bitcoin illegal zu machen. Dies ist nicht nur ein Softwarefehler, sondern kann sogar als struktureller Fehler von Bitcoin angesehen werden. Schließlich sind die freie Teilnahme und die Unzensierbarkeit von Bezahlungen Attribute von Bitcoin, die viele Benutzer als essentiell erachten, aber durch diese Möglichkeit der Illegalisierung wird ggf. das gesamte System unter Druck gesetzt oder abgeschaltet.

Diese Thesis macht einen zweistufigen Vorschlag, um das Problem zu lösen. Durch die ersten Stufe wird eine effektive Dateninklusion für einige Standardtransaktionen verhindert. Dadurch wird es Netzwerk-Teilnehmern erlaubt, mit Sicherheit auszuschließen, dass bestimmte Transaktionen potentiell illegale Daten enthalten. Sie können diese Transaktionen also problemlos verbreiten und in Blöcke schreiben lassen. Außerdem verhindern wir so, dass die Ressourcen von Bitcoin-Knoten durch diese potentiell illegalen Daten belegt werden. Bestimmte Teile der Standardtransaktionen müssen nämlich sogar so lange im Arbeitsspeicher gehalten werden, bis weitere Transaktionen sie referenzieren. Dies ist aber bei Transaktionen mit illegalen Daten normalerweise nicht möglich bzw. wird nicht gemacht, um die Datenlöschung zu verhindern. Der Verhinderungsmechanismus für diesen Angriff besteht aus einem System, das Beweise erzeugen und weitergeben kann, die einem Bitcoin-Knoten versichern, dass bestimmte Adressfelder vorraussichtlich gültige Bitcoin-Adressen sind und keine zusätzlichen Daten enthalten.

Die zweite Stufe behandelt weitere Regeländerungen in Bitcoin. Zuallererst sollten alle nicht-standard Transaktionen verboten werden. Eine Ausnahme bilden dabei die flexiblen und populären P2SH-Transaktionen. Dadurch werden die verbleibenden Dateninklusionsmöglichkeiten limitiert, ohne die Flexibilität oder die Smart-Contract-Fähigkeiten von Bitcoin komplett zu entfernen. Des weiteren soll das Layout von P2SH-Transaktionen angepasst werden, um eine fein-granulierte Löschung bestimmter Datenfelder dieser Transaktionen zu ermöglichen. Das soll es möglich machen, die problematischen Teile einer Transaktion zu löschen oder nicht mehr zu verteilen, ohne die Integrität der Blockchain

oder der übrigen Teile der Transaktion zu verletzen. Schließlich wird noch eine spezielle Transaktionsart eingeführt, die genau diesen Teil einer Transaktion als löschbar markieren können soll. Die Einschränkung ist jedoch, dass solch eine Transaktion genausoviele Bitcoin bereitstellen muss, wie vorher durch den P2SH-Teil bereitgestellt wurde. Dies verhindert verschiedene Gelderzeugungsangriffe auf Bitcoin, und ermöglicht trotzdem eine Löschung von illegalen Daten in der Blockchain, wenn diese von einem beliebigen Bitcoin-Nutzer gewünscht wird.

Durch diese Änderungen können illegale Daten entweder erst gar nicht in die Bitcoin-Blockchain gelangen, oder die Daten können aus der Blockchain entfernt werden, ohne die Integrität des Bezahlsystems als Ganzes zu gefährden. Die Zensur betrifft also lediglich bestimmte Transaktionsdetails, jedoch nicht die Bezahlung selbst.

Die komplette Implementation dieses Vorschlages würde den Umfang einer Masterarbeit leider sprengen. Stufe 1 wurde jedoch in der bekanntesten Bitcoin-Node-Software, Bitcoin Core, als Proof-of-Concept implementiert. Stufe 2 hingegen wird ausführlich erklärt. Außerdem werden Hinweise zu potentiellen Auswirkungen dieser Änderung gegeben.

Des weiteren geht es in dieser Thesis um frühere bzw. andere Methoden, welche Daten in ähnlichen Systemen löschbar gemacht haben. Die Behandlung dieser Methoden soll dazu führen, dass die präsentierte Lösung und ihre geringeren Auswirkungen auf das Bitcoin Ökosystem besser verstanden werden. Andere, extremere Maßnahmen haben unter Umständen nach schlechte Chancen, in Bitcoin aufgenommen zu werden. Das System würde durch sie wahrscheinlich stark zentralisieren werden. Dies steht im Widerspruch zum Grundgedanken von Bitcoin, das eine hohe Dezentralisierung anstrebt, um eine Kontrolle des Netzwerkes durch einzelne Personen oder Staaten zu unterbinden.

Contents

1. Introduction

1.1 Motivation

With over two thousand citations of the original Bitcoin paper [Nak08] by the pseudonymous author Satoshi Nakamoto, it can be said that the scientific community has accepted Bitcoin and related crypto currencies as a new and interesting field of research. Bitcoin is an open source, community driven software project that incorporates techniques previously seen in other distributed networks, and that heavily rely on cryptography. The primary innovation is its use of "Nakamoto consensus", a consensus mechanism that allows the network to obtain a single, shared view on the order of transactions. It is sybil resistant without requiring access control or any kind of central authority that could present a liability if controlled by a powerful actor, and the method of operation that enables these guarantees allows every participant to guess the likelihood of a full compromise of the system by nefarious actors.

These qualities have attracted IT professionals and critics of national monetary systems alike. On top of that, speculation and stories of people becoming millionaires has cause a huge run-up in popularity in 2017. Prices for single tokens of cryptocurrencies have also risen way above anything seen prior to 2017, and even ransomware has adopted Bitcoin instead of other payment possibilities, probably because the worldwide availability of cryptocurrency has increased to suitable levels. Bitcoin is used as exchange currency, meaning people send it to people all around the world over the Internet instead of using other providers that charge a potentially higher fee to deliver and exchange currencies, and some companies like Amazon, Microsoft or gaming platforms have started to add Bitcoin as a payment method just like they support credit cards and bank transfers.

1.2 Problem Description

Through this increase in adoption, the security of Bitcoin itself and its dependent technologies such as the software running Bitcoin exchanges and vendor hardware have become an interesting target for exploitation. Besides attackers that want to enrich themselves by stealing bitcoins or causing a denial of service for popular services and demanding ransom, nowadays the potential for hurting the legality of the system has also become a possibility for profit making. This became possible through the introduction of financial tools, such as put options and futures, that basically enable anybody to bet on a falling Bitcoin price. Through this, an effective DoS or just the fear of a DoS might enrich the attacker, if they

previously used those tools to bet on a price decline, just like rigging a bet in a sports tournament by paying one party to lose intentionally.

This reason, among other reasons such as energy consumption and general problems of creating secure software, but also potential side effects and legality problems hinder a faster and more widespread adoption of Bitcoin. This thesis deals with the problem of unlawful or morally objectionable data being inserted into the Bitcoin blockchain by third parties, effectively hurting the availability and thus security of the whole network. As Bitcoin is a permission less system, and because the blockchain does protect data from being changed after the fact, this type of attack to hurt the network appears to be hard to defend against. The state of the art does not offer a viable course of action should the attack be started right now, at least if the important qualities of trustlessness and accountlessness of Bitcoin are to be preserved. Potential solutions using coordinated purges of the blockchain or consensus mechanisms that control the deletion of Bitcoin history would all be ultimately abusable by powerful attackers, and affect the integrity of Bitcoin as a means to facilitate electronic payments. Other theoretical solutions that replace the blockchain with e.g. account balance commitments may solve the requirement of storing old blocks entirely, but at the expense of trustlessness of the payment system, as they lack some guarantees of Bitcoin, namely that the system has always operated according to its rules when checking the validity of transactions.

1.3 Approach

The core idea to solve this problem is to disallow the inclusion of non-provably unproblematic data for simple payments. If we have a system that can verify proofs that a simple transaction is free of potentially harmful data, and this system still allows for payments to be sent and received, further solutions can be built that utilize these transactions to remove more complicated transactions where data inclusion is still possible. Namely, our second part of the solution requires payments to remove more featureful blockchain transactions that could contain immoral or illegal data. This allows us to preserve most of the capabilities of Bitcoin, while only removing the quality of unchangeability in those cases where it is actually needed to allow for the removal of potentially harmful content. This change to Bitcoin does not endanger the rest of the system by causing the basic security assumptions to change, and the only current Bitcoin dependent technologies that will not be possible after implementation will be services that require arbitrary data to be savable to the blockchain. This use case obviously cannot be combined with a system that should somehow allow the removal of data, however. That said, the idea presented here does not require every Bitcoin participant to remove data that was deemed offensive or illegal, but leaves the decision to the user.

1.4 Contribution

The contributions of this thesis are an analysis of the problem, including the attempts to solve it within similar systems and cryptocurrencies. Arguments for their viability or their suitedness in the case of Bitcoin are provided. Additionally, a proof-of-concept implementation of the part of the solution that allows nodes to be sure that no arbitrary (and thus potentially illegal) data has been included within a transaction is meant to show that there exist no large practicability issues that might prevent the solution from being implemented. This implementation is based on a recent version of the most popular Bitcoin node software that was started by the founder of Bitcoin himself, a choice that was meant to further support the argument of the viability of the proposed changes. As a proof-of-concept, the implementation does not update some less critical features e.g.

at the network level, since their implementation would require significant development efforts beyond the bar set for a master thesis. Such an implementation, however, would be necessary to be provided in case of real-world deployment.

1.5 Outline

After this introduction, this thesis continues with a thorough explanation of relevant parts of the Bitcoin node software and protocol [2].

Then, general related work covering Bitcoin will be presented [3], moving closer towards those papers that aim to address similar problems as those discussed in this work.

Following the related work, potential solutions and their shortcomings or inability to stop illegal data inclusion are discussed [4.1]. They range from data detection to partial or full deletion or schemes that might be not applicable to Bitcoin because of centralization issues.

After specifying what part of the problem will be solved and which details might not be covered by the concrete implementation presented in this thesis (and why that is not a problem) [4.2], the solution will be thoroughly explained [4.3]. The solution chapter deals with the proposed additions to the protocol to stop unwanted data from making it onto the blockchain. A proof system will be described that allows transaction creators to convince block creators that their transactions are free of any arbitrary data. Furthermore, changes to the blockchain data structure will be discussed that will retain the possibility of non-standard transaction usage via the safe deletion of certain parts of a block [4.4]. The downsides of the approach such as the potential for censorship or incentives for attacks will also be discussed.

The proof-of-concept implementation will be presented in chapter 5 with an explanation on how to run the implemented commands to be able to test and expand upon the scheme [5.2]. Finally, some implementation details about the proof of concept will be shared with the reader [5.3]. The thesis ends with a conclusion [6], summarizing the proposed solution and highlighting some potential future work.

2. Background Information about Bitcoin

Giving insights into current developments surrounding Bitcoin requires the reader to have a basic understanding of the protocol rules. Furthermore, some details that are not visible to a casual user of Bitcoin can be very important to understand why some proposed changes are even possible. For this reason, this document gives a general high-level overview of the functionality of Bitcoin, while explaining some details (e.g. of transaction validity, or scripting possibilities) that will be useful when trying to understand the opportunities and challenges described later on.

2.1 Public Ledger

As its core functionality, Bitcoin provides a publicly accessible ledger, tracking amounts coming and going to addresses. Unlike a traditional ledger, for example a book used in a bank to track payments going in and out of accounts, the Bitcoin ledger does not immediately store address balances. Instead, it operates on the principle of always referring to previous transactions and their amounts, and combining them to create a new transaction addressed to a different recipient.

To add more detail, transactions are usually made up of two parts: A list of inputs and their amounts, and a list of outputs plus amounts. The outputs role in a transaction is clear: They specify how many Bitcoins will be sent to which receiver, and enable the sender to pay multiple recipients at once with a single transaction. On the other hand, the role of the inputs is to specify which bitcoins that were received in the past can be redistributed by the outputs. In order to do that, inputs are basically references to previous outputs on the blockchain that the sender can prove ownership of, and that have not been used before to fund a different transaction.

If a previous output used as an input in a new transaction had a higher amount than all outputs in the current transaction combined, using just this single input is fine and preferred. If it does not suffice, more inputs have to be added to the transaction, until the amounts in the inputs match or exceed the amounts in the outputs.

Transactions that try to spend more Bitcoin in their outputs than they have referenced in their inputs are invalid and will be rejected at every point in the system. Similarly, inputs that have already been referenced and spent before fully invalidate the transaction, as does not being able to prove ownership of the previous outputs via Digital Signatures. As one cannot refer to the same transaction output twice, new transactions always use

the full amount specified for them. In order to not overspend or lose Bitcoin when only a smaller amount is to be sent to a recipient, two outputs will be created: The first one will refer to an address of the designated payment recipient and the amount that is to be paid to them. On the other hand, the second address will be an address owned by the original sender of the transaction, containing the surplus bitcoin that were referenced in the inputs and not completely spent through other transaction outputs. As both outputs have not been referred to yet (as they were newly created under a new transaction ID), both the recipient and the sender can now refer to their respective output if they want to use the Bitcoin they received.

Bitcoin cannot be accidentally lost through this mechanism: If we spend more Bitcoin than we have, the transaction becomes invalid, and can never be referenced. If we spend less Bitcoin than we put in, the remaining unspent Bitcoin have to be claimed by creators of new blocks as a transaction fee, a detail of the underlying Bitcoin block creation protocol.

2.2 Digital Signatures and Bitcoin Addresses

If every user could just refer to any transaction output and use its amount in their own transactions, owning Bitcoin would be impossible, as transactions are not kept secret but broadcast to the whole system. This is why Bitcoin proves the eligibility of someone trying to spend an output through cryptographic digital signatures. Cryptographic signatures require public/private key (so called asymmetric) cryptography to function.

Bitcoin addresses play a key role in this. Instead of being numbers given out by a centralized mechanism, only used to address recipients of money, Bitcoin addresses are the starting point for a public/private key signature scheme that allows to verify transaction output ownership. To create an address, the software creates a 256 bit random number and derives an ECDSA private key from it. Using this key, a public key can be created. In early Bitcoin implementations, this public key was used like an address, and users had to share their public key to receive bitcoins. However, the uncompressed ECDSA public keys used by Bitcoin are 512 Bit in size, resulting into a long address if encoded in a human readable format. Even worse, if a flaw in ECDSA was discovered, Bitcoin funds could potentially be stolen if someone could calculate the private key through the publicly known ECDSA public key.

To mitigate this, a new Bitcoin address format and "Pay-To-Public-Key-Hash"-Outputs were created. The resulting string (for example 1Et8vnKhXjYYcz7uQDVVEyo8f5CytE9S3K) is a BASE-58 encoded hash of the public key. If this address format is used to send funds, the public key may remain secret at first.

However, this changes when the coins sent to an address are to be spent in a new transaction. In order to confirm ownership, the sender must provide two things: A public key that, when hashed and encoded in the way described above, matches the address used by the output. On top of that, a valid signature created through the secret key corresponding to the public key, and thus, the right bitcoin address, is required. As the signature can only be validated via the public key, it would not suffice to release the signature only.

The signature plays an important role. The Bitcoin ledger and messages on the p2p network are considered public, so other people can see and potentially copy a valid signature. If the signature would only need to be valid, everybody could copy and try to spend the output until a transaction wins the race of being included into the public ledger.

For this reason, the signature does not only need to exist and be valid, but also contain (and thus sign) a hash of the important parts of the just created transaction, with all its inputs and output (= receivers). This way, the signature can be copied, but is only valid

for exactly the same transaction as the owner intended, making the redirection of funds impossible.

More information on how the validity of Bitcoin transactions is decided can be found in the section "Bitcoin Script". The described functionality actually only covers one type of standard transaction, but the implementation allows for more complex validation rules with a non-turing complete, stack-based language. In 2015, this type of transaction made up about 89% of all blockchain transactions [tra15].

2.3 The Order of Transactions and the Blockchain

While it is possible to prove ownership of bitcoins by utilizing public/private key cryptography, a missing piece of the puzzle is how to decide which transactions happened before other, similar transactions.

Specifically, if someone knowing the secret to an address were to sign two transactions spending the same funds, it would be impossible to determine which one of them should be seen as invalid. Only one can be valid, however, because all others would be spending the same funds that have already been used up through the first transaction. Timestamps cannot be used, as there is no time authority that could be used. One could easily lie about the date when the transaction was created. On top of that, announcing a transaction earlier does not create any guarantee that it will arrive earlier than a second transaction sent later, as the first transaction could be stored and forwarded later. If the second transaction reached their target (i.e. the Bitcoin nodes) normally, the second transaction would arrive first. If the same happened only for some nodes, but not for others, an indisputable decision about which transaction came first could not be reached easily.

To solve this problem, valid transactions are not automatically treated as having happened once they reach a node. Instead, they are put into a list of unconfirmed transactions, waiting to be included into the so called blockchain.

The blockchain consists of a series of block headers that each reference a previous block header by including its SHA256 hash in their own header. A later change of a previous block header would break the chain, as the reference to the previous block is this hash, and if it changes, the reference will point nowhere. If the reference in the following block were to be changed to match the new hash, the error would propagate, as changing the block header of the following block will cause its hash to change, too, invalidating the next blocks reference (and so on). Only a full recreation of all block headers starting with the changed one would allow someone to recreate a working blockchain with the intended changes.

So far, we have not described how Bitcoin stops participants from this recreation of the blockchain from a certain point onward. In the original Bitcoin paper [Nak08], Satoshi Nakamoto describes a so-called "Proof-of-Work" mechanism that prevents just this: By requiring every participant who wants to create a new block to solve a computational puzzle, an attacker "would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes."(p. 3)[Nak08]. Honest nodes are Bitcoin network participants that follow the protocol, meaning they only try to create new blocks on top of old ones. Furthermore, they follow the blockchain that was most difficult to calculate (by verifying the puzzles), and they discard any blocks that solve the puzzle, but break other rules of the protocol.

The security qualities of this system have been analyzed in various papers. In "The Bitcoin Backbone Protocol: Analysis and Applications" [GKL15], Garay et al proved the vulnerability of this system if more than one third of computing power is dishonest. The Bitcoin

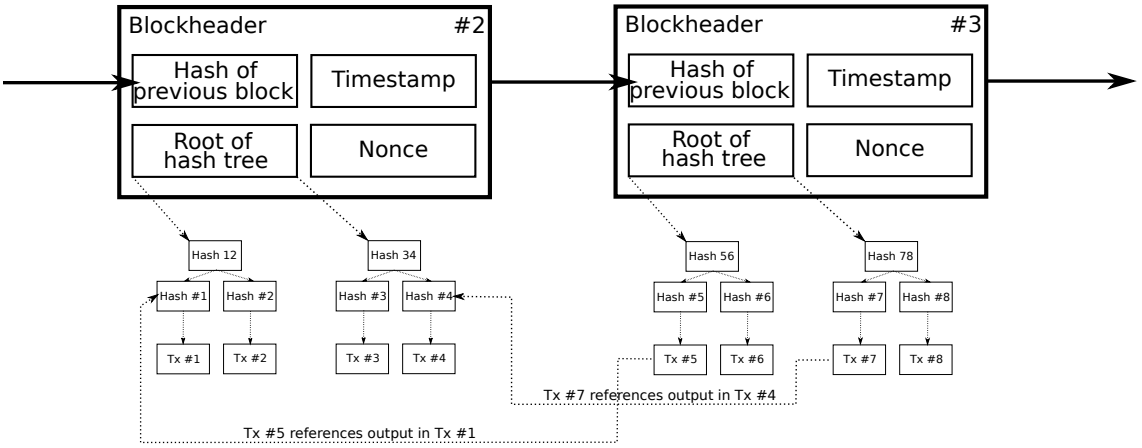

Figure 2.1: Representation of the blockchain.

community deems changes to the system as disputed [Var16], making changes to the protocol only likely if they do not introduce any additional downsides. As this document tries to focus on a different rule change proposal for Bitcoin, it will not go into more detail about proposed changes to the proof-of-work system.

To securely order the transaction with the help of the blockchain, each block contains the root of a transaction hash tree (also Merkle tree). This basic data structure stores transaction IDs as leafs, while the nodes are hashes of their children for performance reasons that we will not expand upon. The order of transactions is defined by the following system: Transactions in earlier blocks have happened before the following blocks. Within a single block, the hash tree provides the ordering by defining the left-most leaf node as the first node, and the right-most leaf node as the last (figure 2.1). It is impossible to change, add or remove transactions without the block header's hash changing, so once a block has been confirmed, the order of transactions is unchangeable without changing the block. Through the provided order of all committed transactions, it can be decided if a transaction is to be accepted and committed, or if it is invalid because a similar transaction spending (some of) the same funds has been committed before.

Should a node create the next block in the blockchain, all uncommitted transactions are valid candidates to add to the merkle tree. However, all added transactions may not spend an output that a previous transaction has already referenced before. This is how the blockchain solves the double spending problem, a problem systems before Bitcoin were not able to solve adequately if they did not introduce a central trusted authority.

2.4 Bitcoin Script

Bitcoin can do more than allowing the secure spending of amounts from one address to another. Other possibilities include sending money to a group of people who will have to agree on how to spend the funds, or to anyone who provides a rule set on how they will be spendable, keeping these rules hidden until they are to be spend by a special address format.

The reason for this flexibility is Bitcoin Script, a stack-based, non-touring complete programming language that only supports a limited amount of so-called OP codes. When an output is created that should pay a certain address some Bitcoin, the output does not

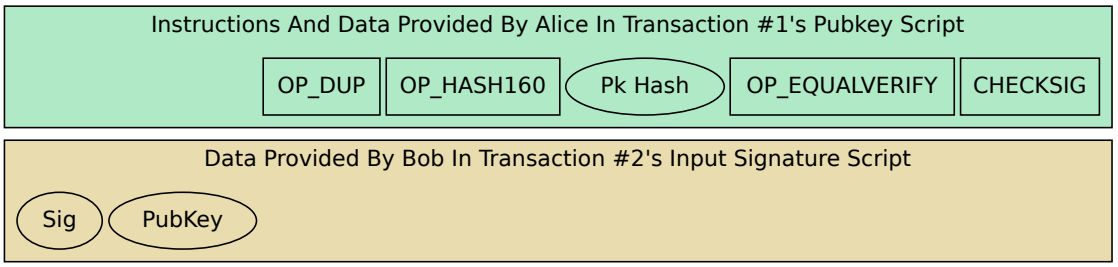

Figure 2.2: A standard output as depicted in the Bitcoin developer reference [bit18d]. Signature and public key are pushed onto the stack. OP_DUP duplicates the public key, then OP_HASH160 replaces it with its hash. OP_EQUALVERIFY then compares the hash to Pk_Hash and invalidates the whole transaction in case they don't match. Lastly, the previously duplicated public key and signature are checked by CHECKSIG, which returns true if the signature is valid for this transaction.

just contain the address and amount, but a short Bitcoin script. This script will have to return 'true' when an output is being spent, or else the whole transaction is deemed invalid, making it impossible to be included in a valid Bitcoin block. As anybody would be able to spend a Bitcoin output where the script returns 'true' when ran without any input, nearly all output scripts only return true when supplied with specific data. The designated way to provide that data is to add it (and potentially additional Bitcoin script code) to an input when trying to spend the output. The data is then put at the front of the script, and then the whole stack of opcodes and data will be ran by an internal scripting engine, and return either true, or false, depending on the supplied data.

With a standard pay-to-public-key-hash output (= a Bitcoin address with a leading 1), the Bitcoin script consists of only a few functions ran from left to right. Figure 2.2 explains this in detail. Any type of script (within certain size restrictions) can be written, including erroneous ones that never return true.

Covering the scripting language in more detail goes beyond the scope of this document. The important takeaway is that the system provides some flexibility, including conditional statements and comparison checks for the supplied data. This flexibility also allows for inclusion of arbitrary data, which causes a part of the problem that we want to defend against with the solution that is presented in this document.

2.5 Mining and Initial Coin Generation

Instead of creating an initial transaction that creates all Bitcoin and having the creator of Bitcoin supply those coins to those who want to be part of the system, Bitcoin opted to reward nodes that create new blocks with newly created coins. When a new block is created, the first transaction has to be a so-called coinbase transaction. These transactions have no input and as such no prior coins that determine the amount, but still send a certain total amount of Bitcoin to one or more outputs. This amount, which essentially creates Bitcoin from nothing, started at 50 Bitcoin, and will continually halve every few thousand blocks until 21 million coins have been created. At the time of writing this, the so called block reward is 12.5 bitcoins, and the next "halvening" will probably occur in 2022 when about 94.000 more blocks have been found, resulting in the 630.000th Bitcoin block having been created.

Distributing the total supply of all bitcoins between supporters of the blockchain has the benefit of decreasing the amount of trust one has to put in the creators of the software, as they do not exert control over the complete monetary supply. On top of that, the creation of blocks is able to occur decentralized, because people are incentivized to create new blocks, meaning people already interested in Bitcoin have a further incentive to support the network by running the software to create new blocks.

Bitcoin mining is essentially the creation of a partial hash collision for SHA256. The found hash has to also be a hash of a valid Bitcoin block header, and this header is required to be the header of a valid Bitcoin block.

This means there are essentially two kinds of work that are required to successfully mine bitcoins: First, a valid block has to be created, potentially including many new transactions, and conforming to all rules set forth by the consensus of the network and implemented in popular node software. Second, a stable supply of energy and proper hardware for SHA256 hash collision creation has to be sourced and needs to be run, ideally without any interruption.

Mining pools are a development where people specialized on block creation provide a highly available service to which to provide hashing power to. According to this hash power, payouts will be done when a bitcoin block is found. This means a miner can concentrate on the second part of Bitcoin mining, if they trust the pool operator to mine valid blocks that pay the person owning the mining equipment a fair share.

The incentive to join a pool is quite high especially for owners of only a small share of total hashing power. As the difficulty adjustment algorithm of Bitcoin causes the network to only find about 144 blocks per day, it is clear that it might take a very long time for a single miner to earn back their power and hardware investment if there are more than a few thousand miners trying to find blocks. However, a pool where many miners combine their hashing power will find blocks on a more regular basis, and the block reward can then be split between all participants according to their contribution to the pool. Counting the contributions is especially easy to accomplish, as a pool can use supplied shares, meaning partial hash collisions that aren't good enough for the Bitcoin network, to calculate how much hash rate has been supplied.

As pools essentially control the process of block creation, they can present a problem for Bitcoin. Censorship of single transactions is more likely if only a few pools and no single miners exist, as it might be possible to force the pools to not include certain (valid) transactions. Additionally, a pool controlling over 50 percent of the total hash rate could theoretically earn all future mining rewards and split it between its supporters, as blocks created by other pools could just be disregarded while their own blockchain would still present the fastest growing blockchain that is deemed the proper one by all Bitcoin clients.

Pools may also have positive side effects, however. They are experts on Bitcoin consensus rules, block creation and node software, and can thus advise their supporters on what Bitcoin updates they should consider. In the event of a problem with block creation, they also serve as central communication points for Bitcoin developers, such as in the 2013 chain-split event where a downgrade of the Bitcoin software prevented the network to fall victim to a potentially permanent chain split [Nar15].

For the purpose of the changes discussed in this thesis, pools could serve as proponents of the idea to keep their operations legal. Unlike hash rate providers, they might be held responsible for illegal content on the blockchain. A pool is also much harder to operate anonymously compared to a single mining operation. As popular attack targets for denial of service, they have much larger bandwidth requirements compared to a miner

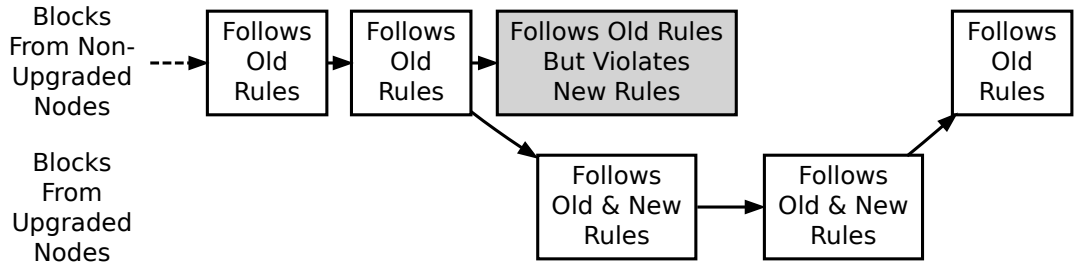

Figure 2.3: Forking as depicted in the Bitcoin developer reference [bit18b].

that essentially only needs to download new blocks, and who can publish a found block through multiple obscured pathways.

2.6 Chain Splits

A topic that should be covered in light of proposed updates to Bitcoin are network splits. One such split may occur when the system is updated, and is called a hard fork. It should not be confused with forks in open source projects where code is copied and worked on by a different group with a different name, although they might coincide.

A fork on Bitcoin happens when a single node or a group of nodes create new blocks that are not valid under the old rules. The network then splits into two groups, one that builds new blocks on top of the block with the different rules, and one that chooses to not do so and build on top of the most recent block still following the old rules.

Chain splits are different as they can also occur without any change to the Bitcoin code base. They might be caused by a split in network connectivity, making some nodes unaware of newer blocks being created elsewhere on the internet. In this event, multiple groups that do not see all other groups' nodes would work on their own blockchains. When the network connectivity has been restored, only the blockchain with the most accumulated proof of work, e.g. hashing power invested, and thus longest blockchain will be considered valid. The process of ignoring shorter blockchains that were created by the other groups is called orphaning. In case of similar hashing power being invested, it may not be clear which blockchain will win this race. Eventually one blockchain will have found an additional block before the others, however, and the system will converge onto a single blockchain view once again.

As this thesis discusses rule changes for Bitcoin, chain splits and forks are a possibility. The changes discussed in chapter 4 will essentially cause a slightly different kind of fork as rules will not be changed complete, but only made stricter, as depicted in the above figure[2.3]. In our case, old nodes can continue validating new blocks, but they will be unable to create new ones as the stricter rules require additional information that un-updated nodes will not know how to provide. As such, new blocks may automatically violate the new rules, and have their blocks abandoned if the majority follows the new, stricter rules.

The concrete details of a fork or chain split because of the new rules are not a focus of this thesis, as they are unlikely to be different from other, previous changes that behaved similarly. One such change is segregated witness, or SegWit, which we will introduce for another reason in the upcoming section.

2.7 Segregated Witness

In 2017, a big update for Bitcoin went live. The code change had multiple different goals [Tow16]. Its most prominent change to Bitcoin was seperating output signatures into an extra data structure. This segregation of the witness, i.e. the data that proves that a transaction is valid, gave this update its name. Segregated Witness, or SegWit, took multiple years to design, implement and activate. We will first discuss all changes and aims to give an overview and a general understanding of Bitcoin goals and implementation aims, and then focus on implementation details that are relevant to the proposed solutions presented in this thesis.

2.7.1 Malleability Fix

Certain parts of Non-SegWit Bitcoin transactions can be changed by third parties, resulting in the transaction gaining a different ID. This is only a problem before the transaction is included inside a block, as the merkle root will then commit the transaction to its current form. However, before a transaction is added to a block, the ECDSA signature or the ScriptSig can be changed. In case of the ECDSA signature, Bitcoin nodes historically accepted various encodings of the same signature, as the library openssl was more lenient on encoding rules than the standard itself [Eno11]. Some fields of the utilized DER encoding can even be changed arbitrarily, causing a different transaction ID for every such change.

Similarly, various opcodes can be added on top of a valid ScriptSig even after a transaction has been created and signed. As long as the opcodes do not change the validity of the transaction, such as if they push something to the stack and then remove it, this changed transaction could be added to a valid block, just as the original one. When the changed transaction is committed, the ID of the transaction will be different than the expected ID the sender calculated. This can cause problems when implementing third party software that relies on Bitcoin transaction IDs.

It is possible because the signature contains and signs a hash of the data that should be secured. However, the SHA256 hash function requires the complete input string that is to be hashed before it can run. Hashing a string that is meant to include its own output is not possible. It would create a circular dependency, where the SHA256 function needs to know its own output before it has been run. For this reason, Bitcoin's transaction signatures do not include the ScriptSig. The downside of this approach is that modification of the ScriptSig by third parties, such as adding more opcodes, is possible. On the other hand, the main transaction ID does include the ScriptSig. After all, the ID is used to verify the integrity of the whole transaction, and not adding the ScriptSig in this case would enable an attacker to add data to make the verification fail, potentially invalidating the block for other users.

The fix to malleability presents a small improvement that might prevent sophisticated data inclusion attempts, which is the reason for its mention here. The malleability fix was also one of the main reasons for the creation of the SegWit update, as it allows for payment solutions that do not require every transaction to be saved to the blockchain. This obviously circumvents the problem for many transactions, as a transaction that will not end up on the blockchain is not a good target for illegal data inclusion.

2.7.2 Future Expandability

Adding new features to Bitcoin without requiring the whole network to upgrade was possible even before SegWit. Satoshi Nakamoto added placeholder opcodes (NOPs) to Bitcoin whose functionality can be changed for newer versions. If an older client sees such an opcode, evaluation of the NOP will cause nothing to change, essentially allowing the rest of

the Script to decide about the validity of the transaction. On the other hand, newer clients could run more elaborate checks when they encounter the reassigned NOP codeword, and if the check fails, invalidate the whole transaction outright.

As one can see, this update mechanism is inflexible. The rest of the transaction has to cause the validation to succeed, so the new opcode should not change the stack in a way that makes it valid, else old nodes will deem blocks containing such a transaction as invalid, breaking backwards compatibility.

SegWits resolves this downside by handling specially formatted Bitcoin Script in a different way. If a script conforming to the layout and of a specific length is detected, the software runs SegWit specific validation rules. These can be completely arbitrary and do not have to be implemented with Bitcoin Script. Still, Bitcoin Script is also usable for such new transactions, as it enables the implementation of Smart Contracts by Bitcoin users without requiring widespread consensus and code changes to the main Bitcoin project.

This new SegWit output format includes a version field. The field will be used by future Bitcoin updates to denote new transaction types, only to be evaluated by newer software that knows how to validate them. Compared to that, older software will see a version that they have no validation routine for, and just deem it valid without further validation. Compared to ignored opcodes, rules can remain simple and short, as old node rules do not need to be taken into consideration when designing scripts and their ScriptSig counterparties.

This new possibility for updates could also allow the discussed proposal in this thesis to become implemented in a less contentious way. By only enabling the later discussed proof requirements for newly defined SegWit transactions, old systems would not require any update. It would not protect any participant from the problems discussed, however, as Bitcoin would remain just as vulnerable as before if the old transaction types were not equally updated. Still, these additional capabilities for updates introduced with SegWit might provide some use. For instance, a SegWit specific update could be seen as a first step until full agreement by users and miners can be reached. If the system works as designed on the real network, a later update could then extend the rules to the remaining transaction types.

3. Related Work

Bitcoin was presented to the world by the pseudonymous author Satoshi Nakamoto by uploading a scientifically styled paper to the cypherpunk mailing list metzdowd.com on the last day of October in 2008. It contains a total of eight references, and those either deal with previous proposals for an electronic cash system, or scientific papers whose results might be considered building blocks for the solution Satoshi Nakamoto has devised. In the following chapters, we will cover some more general references to Bitcoin and electronic cash, and then focus on papers that have to do with similar problems to the one covered in this thesis.

3.1 General Research in regard to Bitcoin

3.1.1 Before Bitcoin

In the whitepaper, a text from 2002 by Wei Dal titled bmoney.txt is cited as an inspiration for the basic idea of requiring the transmission of every transaction to all participants of the payment network. Wei Dal also saw the need for a consensus based money creation scheme that is independly verifiable by new participant, but his protocol requires a penalty system that punishes servers committing to wrong account balances or inflated monetary supply. To solve this, Bitcoin adapted an idea described by Adam Black in a paper titled "Hashcash - A Denial of Service Counter-Measure". In his paper, Black formulates a cost-function that allows the receiver to verify that a certain amount of hard calculations must have occurred without needing to do an equal amount of work to verify it. He explores variants of this idea, and explains the up- and downsides of possible protocol implementations. Following Blacks research, Bitcoin would be categorized as utilizing a publicly auditable, trapdoor-free, unbounded probabilistic and interactive cost function that effectively changes the server challenge difficulty by modifying the parameters according to a Bitcoin specific algorithm. Unlike described in Black's paper, the (proof of) work is not done on a predefined identifier such as the name of the service, but the hash of the as of yet unannounced block of the blockchain.

Other papers cited in the Bitcoin document describe basics for public key cryptosystems [Mer80] or probability theory [Fel08] as Bitcoin utilizes them, or makes assumptions based on probability theory to argue for its security in real-world attack scenarios.

Besides these papers, Bitcoin today utilizes many different techniques that have been released with accompanying papers or that were later analyzed in such.

3.1.2 After Bitcoin

After Bitcoin was released, researchers did not take note of it immediately. The first widely cited papers that deal with cryptocurrency or Bitcoin specific attributes appeared in 2011. Papers on Bitcoin often cover vastly different parts, such as (world) economics, incentive systems, distributed systems, anonymity, cryptography, computer security and papers focusing on concrete technical implementations of the Bitcoin system.

One of the earlier and often cited papers by Reid, Fergal and Harrigan [RH11] deals with the lack of anonymity on the Bitcoin blockchain, allowing an attacker to combine various payments to at least pseudonymously track future and past payments of the same entity. Another paper titled "Analysis of Bitcoin pooled mining reward systems" [Ros11] deals with technical implementation details concerning Bitcoin mining. Pooled mining is a system where participants do not solve Bitcoin blocks individually, but combine their efforts to solve the problem of probabilistic reward distribution. As it is now, the algorithm targets 144 Blocks to be found each day, which means that in the event of tens of thousands of miners mining Bitcoin, finding a block and thus receiving the mining reward which is essentially the payment of miners might come too late to pay for electricity and hardware requirements. This development hasn't been anticipated by Satoshi Nakamoto at the time of Bitcoins creation, but imposes new problems of reward distribution and attack vectors for Bitcoin, which is what the paper by Rosenfeld [Ros11] sheds light on. After the rise of the Bitcoin price beyond one dollar per bitcoin in 2011, economy researchers also started to investigate Bitcoin more thoroughly. In "Bitcoin & Gresham's Law - the economic inevitability of Collapse" [GG11], Güring and Grigg apply economic models on Bitcoin and predict that stolen electricity and botnets may drive out honest participants, eventually collapsing the Bitcoin system. While these predictions have not become true as of now, it is widely observed that mining activity in Bitcoin moves to places where cheap energy can be acquired.

A real influx of important papers started in 2013 and 2014. "Majority Is Not Enough: Bitcoin Mining is Vulnerable" [ES14] by Eyal and Sirer is an often-cited paper that deals with a central security assumption of Bitcoin. Unlike first presumed, an attacker does not need to control more than fifty percent of the hash rate to be able to create more Bitcoins than their hash rate should allow. The results call for more decentralized mining operations than can be observed on the network right now, and explain how keeping miners below 25 percent of the total hash rate should not allow them to do the described selfish mining attack.

3.1.3 Bitcoin Improvements by the Scientific Community

Even though Bitcoin utilizes well-known cryptographic methods, this does not mean that new developments aren't picked up by Bitcoin. In some cases, novel cryptographic ideas are even created with Bitcoin in mind. For instance, Gutoski et. al. presented a paper [GS15] that describes an advanced, deterministic Bitcoin wallet where some private keys may be leaked without the whole wallet becoming compromised by the attacker. The use case for deterministic wallets is not completely dissimilar from some other problems that can be solved with cryptography, but apparently, some cryptographic features are not needed in many other circumstances, and Bitcoin gave a good-enough incentive for scientists to work on an improvement.

These examples show that Bitcoin development can be based on scientific research. In some cases, research and novel solutions are created because of Bitcoin. It can thus be said that Bitcoin is an interesting real-world occurence that utilizes cryptography. It drives further research and requires researchers of various specializations to allow for a more exact understanding of Bitcoin's operation and future development.

3.2 Research into Non-Eternality

Besides the more general papers on cryptocurrencies, there has also been research in areas more closely related to this thesis' topic. Making computer systems forget information that they stored as a byproduct, or even previously relied on is not an easy task, but one that benefits from analyzing the problem thoroughly.

3.2.1 Problem Analyzation

In their paper "Cryptographic Currencies from a Tech-Policy Perspective: Policy Issues and Technical Directions" [MLS+15], McReynolds et al. describe various issues surrounding the use and implementation of cryptocurrency within current jurisdictions. As its purpose is that of an overview paper, they not only cover the problems of anonymous payments, taxes, or theft, but also that of illegal content in the Bitcoin blockchain. They correctly note that there have been instances of images and non-transaction text is stored on the blockchain, and that a malicious party could intentionally store illegal files in it. Unsuspecting miners or users of Bitcoin might then be held accountable for storing and transmitting those illegal files. They go on that this could cause cryptocurrency to even become illegal if someone wants to store every part of the blockchain on their own system, and that different jurisdictions around the world might only cause this to become illegal in certain countries. In their paper, they discuss the possibility of a future blockchain software that introduces a central agent that is allowed to change blocks while retaining their transactional results. While this would be antithetical to some underlying Bitcoin principles, they leave it "as an open problem whether it might be possible to achieve these properties in a new cryptographic currency" [MLS+15].

3.2.2 X-pire!

This thesis is meant to present an update to the Bitcoin protocol that serves as a first step to make this possible, but without introducing a new central authority. The notion to remove content from non-centrally controlled systems like the internet is much older than Bitcoin, though. In their paper "X-pire!-a digital expiration date for images in social networks" [BBD+11], Backes et al. explain a system that allows users to make images unviewable after they have been transmitted to third parties. Similar to the problem of deleting bad data from the blockchain after it has been discovered, this system is meant to make data unavailable when it is found to be problematic. In the case of X-Pire!, only pictures are protected, but in a way that preserves the data format so that standard picture uploading services can be used. They embed information about the encrypted content into the image, and through this information, the decoding keys can be obtained from a third party server. In the case that a picture should not be viewable anymore, the server does not disclose the key anymore.

3.2.3 Vanish: Increasing Data Privacy with Self-Destructing Data

A similar technique is described in "Vanish: Increasing Data Privacy with Self-Destructing Data" by Geambasu et al [GKLL09]. In their case, however, they split up the needed decryption key into multiple chunks and have those be distributed by a peer to peer DHT network. An interesting property of the system is the automatic inaccessibility of data that is no longer needed, at least as long as many peer to peer members behave properly. While a mixture of these both systems might solve Bitcoin's problem of illegal data on the blockchain on first thought, the requirement for one or more third party servers of the first paper centralizes trust and power. As decentralization is one of the prime attributes of the Bitcoin network, it would be unlikely to see it implemented as a solution for Bitcoin. Even if the idea of the second paper were to be properly utilized [WHH+10] to decentralize the

system a bit more, that peer to peer network would not necessarily be part of the Bitcoin network, and thus operate with different incentives. Furthermore, the problem for Bitcoin is the removal of intentionally placed illegal content. While it might be possible to make the encryption key non-choosable by an attacker, so as to not allow it to be effectively zero and thus known by everybody, this system functioning as intended is actually a security issue for Bitcoin. Bitcoin depends on the ability of every user to check transaction and block history for themselves to be sure that no rules were broken. If even just one transaction is not verifiable, this can potentially be abused to inflate the monetary supply or to steal Bitcoin in certain cases. Considering the whole picture and important attributes of a decentralized and trustless cryptocurrency, these new attack vectors might be considered unacceptable by many. In turn, this motivates the search for another solution with less severe drawbacks.

3.2.4 µchain: How to Forget without Hard Forks

One step into a potentially useful direction is "µchain: How to Forget without Hard Forks" [PDC17] by Puddu et al. Their solution removes the immutability of the blockchain by introducing consensus based allowances for a selectable entity or group to create updated transactions that contain different smart contracts or transactions. This could potentially allow for changing contracts or monetary transactions after the fact, which in turn could allow for monetary inflation of the blockchain tokens (bitcoins) in case follow up transactions are not somehow invalidated, or the updating of smart contracts that have vastly different rules. For instance, this might include a smart contract that effectively steals the funds and transmits them to all members of the entity or group that may create update. As this is a security risk with a high incentive for the attacker, the paper discusses some restrictions as to what update transactions may do. One example is the invalidation of transactions that build upon invalidated outputs. However, its first variation creates an implicit additional duration after which a transaction can be considered finalized and after which nobody can remove illicit content again. In the second variation without a time limit, no transaction can ever be seen as finalized, and content insertion could allow to revert and double spend cryptocurrency that previously built upon the invalidated transaction. It is an open question if other solutions might exist that enabling proper patching or content removal functionality using µchain without these drawbacks.

3.2.5 Redactable Blockchain

The concept described in the paper "Redactable Blockchain or Rewriting History in Bitcoin and Friends" [AMVA17] by Ateniese et al. presents a similar solution, albeit with similar problems. They utilize special chameleon hash functions that allow certain individuals to create blocks with the same hashing result, effectively allowing them to exchange full blocks (with potentially illegal data) for updated versions. If the trap door mechanism of the chameleon hash function could be outsourced to miners, and if it could be made impossible to turn this into a way for miners to steal or attack the integrity of the monetary system, it might provide a solution to the problem of illegal data inclusion. However, it is unclear how this can be achieved, as the chameleon hash function in the paper by Ateniese et al. replaces the old block with a new version and effectively calls for nodes' to disregard the earlier block. This means the miners cannot know if substantial or disallowed changes have occurred, and might enable miners or short-lived hashing majority attacks to effectively invalidate important transactions to enable them to enrich themselves.

3.2.6 Analysis of Illegal Blockchain Content

In "A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin"[[MHH+18]], Matzull et al. analyze the problem of arbitrary data inclusion in cryptocurrency blockchains.

They discuss the various methods used to encode data for blockchain insertion, and make a quantitative analysis of currently stored content on the blockchain. Their paper includes an overview of possible ways to insert data and services that might use them, and a Cost per Byte calculation for each method. The methods are sorted into two categories: Those utilizing the outputs scripts inside a transaction, and those that use input scripts. While the former can be thought of as the recipients of a Bitcoin transaction, the later serve as cryptographic proof that all used bitcoins are owned by the creator of the transaction.

First, Matzull et al. describe output methods, starting with OP_RETURN. OP_RETURN is a "controlled channel to annotate transactions without negative side effects" [MHH+18]. They identify the negative side effects, namely pollution of the spendable transaction output cache and note their current limitation to 80 Bytes per transaction.

The coinbase transaction that makes up the first transaction of every block is another vector for data inclusion attacks that was identified in the same paper [MHH+18]. The authors conclude it as an inefficient way to insert data, as only active miners are able to use this method, and the amount of data that can be inserted in total is limited.

On the topic of non-standard vs standard transactions, they describe the inefficiency of using non-standard output scripts because of the difficulty of getting them included by miners, whereas "non-standard input scripts are only required to match their respective output script" [MHH+18], allowing for arbitrary data inclusion "if their semantics are not changed, e.g., by using dead conditional branches". As an example, an attacker could create a P2SH transaction for an input that contains a very simple branch that allows the spending of the output in case they own the proper private key, just as in P2PKH or P2PK. In the effectively dead conditional branch that they never intend to use, they might add a condition that looks like a pay-to-multisig or a similarly large collection of data that might be mistaken for public keys. As it doesn't need to evaluate to true on the stack, there is no real restriction on what the attacker can put here, minus some size limits on the total amount of data. If the user then creates a transaction that successfully spends the output, he can permanently include this data into the blockchain even though it wasn't required. It cannot be removed, however, as this would change the hash and thus the ID of the transaction.

In the realm of standard transactions, Matzull et al. identify four transaction templates, namely P2PK, P2PKH, P2SH and P2MS. For these methods they describe that "The respective public keys (P2PK, P2MS) and script hash values (P2PKH, P2SH) can be replaced with arbitrary data as Bitcoin peers cannot verify their correctness before they are referenced by a subsequent input script. While this method can store large amounts of content, it involves significant costs: In addition to transaction fees, the user must burn bitcoins as she replaces valid receiver identifiers with arbitrary data (i.e., invalid receiver identities), making the output unspendable." [MHH+18].

In P2MS, multiple parties are in control of the bitcoins held in an output, but only in case they cooperate and sign the same transaction with their respective private key. In some P2MS schemes, even a simple majority or other such agreement is enough to spend the output. This is the reason why the author of this thesis disagrees with the stated requirement of burning bitcoins in the paper: If enough of the pushed public keys remain valid and in control of the attacker, while the other public keys are actually placeholders for arbitrary data, the output remains spendable while still containing a good ratio of arbitrary data. After spending the output, the data will not remain in the mempool, however, and purging Bitcoin nodes will be able to purge this data after an initial sync. An attacker might thus prefer the attack described in the paper by Matzull et al., as it affects ever node and not just non-purging ones.

The paper goes on to discuss adverse effects of arbitrary blockchain content. Besides outright illegal content which might make the mere possession and transmission of blockchain data illegal, they list copyright violations, malware, privacy violations, and politically sensitive content as potentially problematic.

The paper goes on to describe detected occurrences of data inclusion on the Bitcoin blockchain, and the techniques that were used. Their results are based on blockchain data up until August 31th, 2017. At that time, only 118.53MB, or 1.4 percent of Bitcoin contained various data. They go on to note that most of the data was found in OP_RETURN outputs, a place where arbitrary data is intended to be put. The described ways of including arbitrary data only make up 0.02 percent of all data inclusion transactions and were "virtually irrelevant" at that time. It is reasonable to assume that the use of alternatives to OP_RETURN inclusions will rise as soon as OP_RETURN is discontinued on the Bitcoin network. On the other hand, discontinuation of OP_RETURN seems irresponsible as long as other, potentially more effective ways for data storage on the Bitcoin blockchain exist, as these techniques often have other problematic side effects on the Bitcoin network, whereas OP_RETURN was designed in a way that makes its use bearable.

3.2.7 A Potential Solution for P2PKH Bitcoin Payments

In the paper "Thwarting Unwanted Blockchain Content Insertion", Matzutt et al. describe an interesting approach to stopping content insertion with transaction outputs. Similarly to already described papers, they motivate it with the potential harmfulness of arbitrary data insertion into the blockchain. Then, however, they analyze the capability of heuristics to stop attackers by monitoring transactions in the mempool. As false positives could cripple the Bitcoin network, they "propose to further restrict the text detector to reject only transactions with more than 5 distinct text-holding outputs (100 B of conent). This way, only short and thus harmless texts can be inserted". They are right in arguing that such texts could already be included via an OP_RETURN output, but it is unclear to us if the Bitcoin network should allow OP_RETURN statements at all considering the high risk of arbitrary content insertion. Especially the potential for watermarking or linking multiple OP_RETURN outputs might cause an attacker to split off his illegal data and include it onto the blockchain, while also making retrieval nearly trivial with straightforward code that scans the blockchain for those watermarks.

Regardless of these details, their content detectors reduced the problem while also keeping false positives low. They note that "...the detector's filtering quality is insufficient for binary files and we expect a wide range of evasion schemes to emerge" however. They go on that "this would imply a poor deployability, as novel evasion schemes result in frequent mandatory updates for all honest full nodes". They conclude that "explicit content detection constitutes a firstl ine of defense against unwanted blockchain content and works for text-based content. Mitigating the insertion of unwanted binary files, however, requires more general approaches."

The next proposal in the paper covers costs of content insertion. The authors argue for a mandatory fee increase for every output found within a transaction, meaning fees are no longer governed by the space the transaction will use in certain parts of the block. This change from the current model, where miners include transactions based on the associated risk of their block being orphaned if some other miner finds a smaller block at the same time, and the fee being paid, is not without downsides. While it describes a (higher) minimum fee, making it interesting for miners to include such transactions even under the old assumptions, it doesn't take into account the ability of a miner to receive the fees of his own transactions. The need to wait 50 blocks until they can spend those coins (now

included in the block reward) is a rather minor deterrent and does not increase the cost for this attack by much. On the other hand, the existence of pooled mining and thus a payout of the mining reward to all participating miners stops the attack from being essentially free. The pool might be cooperative, however, and could plan to permanently damage Bitcoin through the illegal content insertion, in which case they might not be interested in paying miners in the first place. An attacker could also solo-mine a block with bad data, circumventing all described protections of a fee increase as long as he can sell the block reward 50 blocks later and before the illegal data is discovered or before it damages the worth of Bitcoin. Another completely unrelated problem is the usage of Bitcoin in less prosperous regions. If a scheme makes data inclusion exceptionally expensive, this might put special on-chain transactions out of reach in poorer countries, or only enable rich people to attack Bitcoin.

Their final proposal to solve the problem of data inclusion in Bitcoin P2PKH transaction outputs seems to be the most promising. They describe a mathematical scheme where it is computationally infeasible to create arbitrary receiving addresses. Using a commitment, miners can check that a certain hash is not just arbitrary data, and thus reject all transactions that do not provide a valid commitment. Their solution is adaptable to the most common forms of payment on the Bitcoin blockchain, namely P2PKH and P2SH. It does increase transaction size by a non-marginal amount. Outputs that basically consist of a single hash and thus 20 bytes of data now become 112 bytes in length. As inputs are usually the culprits for especially large transactions, this is not as bad as it first appear. They calculated that ”a standard (P2PKH) transaction consisting of one input and two outputs grows from 225 B to 409 B”. It is not uncommon for transactions to include multiple inputs, in which case the increase is even less profound.

The performance need to evaluate commitments before block inclusion or when a block is announced is negligible (0.4ms per commitment on an Intel Q9400, released in 2008). The creators of the paper used comparably old hardware, and if we consider that the size of transactions grows in a similar manner to the increase in performance requirements, a reader might conclude that they even out.

The paper was released during the creation of this thesis. Implementing the solutions presented would further increase the resilience of the Bitcoin network against off-topic data inclusion. The solutions this thesis describes can easily be appended, and do not interfere with the plans of the paper, however.

4. Proposed Solution

This chapter will describe the proposed solution. First, some different, but potentially reasonable approaches will be presented. At the same time, the reason for their dismissal will be stated. The chapter continues with an explanation of excluded problems, i.e. aspects that will be solved through different means or that are not deemed relevant at the current state of the system. Finally, our solution will be explained in detail.

4.1 Potential Solutions and Downsides

There are many half-solutions to the problem of malicious data inclusion, and some changes that appear to solve the problem do not actually solve it after further inspection. In this section, we will discuss some of these solutions, and why they are inadequate.

Simple detection of data inclusion is one example for such a proposal. After all, it seems like a straight-forward approach to solve the problem. If we can detect all attempts to include data in the blockchain, mining nodes can be updated to reject and not-mine any transaction that contains unnecessary extra data. Similarly, newly created blocks that do not follow these new rules could be orphaned (e.g. ignored). If enough miners and Bitcoin users supported the update, the update process would be similar to previous updates of the Bitcoin software. However, it is unlikely that detection of data inclusion will become good enough to be employed for this task. Detection is unlikely to become perfect, and an imperfect detection would be easily circumventable by an attacker by just attempting to trick the public scanning engine in secret. Furthermore, scanning for data would also need to be very fast, as new Bitcoin blocks need to be verified as fast as possible to allow a mining node to securely work on top of the new block without risking their own work (i.e. a new block based upon the previous new block) to be disregarded. If the process was not fast, but slow, miners would lose revenue as they would be forced to mine on top of the older block until the check was finished, or risk mining on top of a block that does include illegal data. The work done on top of the not-most-recent block will probably not cause the miners to earn mining rewards, however, as the valid, but slow to check new block will eventually be regarded as the safest one to mine on top of. When this happens, this blockchain will grow faster than the older chain without the newest block, and the network will only consider this longer and more difficult blockchain as the current Bitcoin blockchain. Additionally, steganography and encryption make it very hard for such schemes to become usable even in the event that they work perfectly in obvious cases. Even encrypting data with a trivial, guessable password will usually make it undistinguishable

from random noise, while still allowing users with the right software to easily extract the data from the Bitcoin blockchain. These and other reasons show that the inclusion of such a detection mechanism would very likely be a futile attempt at solving the problem.

If we rule out the possibility to prevent illegal data to end up on the blockchain by detecting data inclusion attempts, maybe a straightforward solution to remove the data once it has ended up on the blockchain might be feasible. The pruning function of Bitcoin does solve some of the legality problems of running a Bitcoin node once illegal data is included in the blockchain: After a certain amount of new blocks have been mined, all old blocks are removed from the pruning node and will thus not be stored or sent by it again. It does need to receive all blocks initially to check them, however. This requirement encourages other nodes to store and transmit the blocks to their peers, as there exists a small risk of historical blocks becoming unavailable in case too few participants provide them. Similarly, network participation might be hindered from finishing block validation if blocks aren't provided, which would restrict new nodes to join the Bitcoin network until someone can provide these blocks again. Many Bitcoin proponents would likely deem this change from the original idea unacceptable, as Bitcoin relies on merchants and users being able to join the network freely.

It is also not possible to stop every node from storing a block, as that would make the verification of the blockchain permanently impossible for everybody, destroying one of Bitcoin's most important features: It's trustlessness. Additionally, even purging nodes store the set of unspent transaction outputs, as they need those for verification of the incoming blocks. This essentially means that all Bitcoin nodes might be storing illegal data that was placed within unspent outputs. They are usually not transmitting this information to other nodes, but even just the requirement of owning or accessing potentially known illegal data is reason enough to consider this approach as insufficient. Similarly, just deleting blocks on one's hard drive or not transmitting them to other peers does not make the task of running a node without the possibility of legal trouble feasible.

Some more feasible approaches would change key attributes of Bitcoin. One such approach is the use of special hash functions that allow certain individuals to create replacement transactions by empowering them to create hash collisions. The hash functions that make this possible are called chameleon hash functions. If data inclusions are detected within a block, individuals knowing the secret can change (parts of a) block, removing or changing the problematic transaction and/or data. The downsides of this approach, however implemented, are obvious: Instead of creating a decentralized, uncensorable payment network that is controlled by no one in particular, it would put a lot of power into the hands of the individuals that control the chameleon hash secrets. Worse still, even if it was somehow possible to utilize zero-knowledge proofs or some other technique to keep the secret from being known by single individuals, it will probably not be able to stop past participants of the network to abuse this power in the future when they aren't profiteering from the system anymore. Additionally, future key participants of the network, such as miners, are unlikely to be able to receive the secret if previous owners or participants do not want to give it up. Implementing a voting system where votes are distributed according to certain measurable blockchain occurrences doesn't solve it, either: The only attributes that might be sensible to use in that case are coin ownership or found blocks (i.e. calculated PoW). This puts a lot of power into the hands of current miners or companies that have access to a lot of coins, like exchanges. Allowing those powerful groups to gain the ability to arbitrarily censor old transactions puts the network at a far greater risk than it is exposed to today through 51% or similar attacks.

It remains to be seen if some of these solutions will be implemented despite being insufficient or potentially damaging to the system. There might even be incentives for

implementing such incomplete solutions, as some jurisdictions consider what is technically possible when deciding about the legality of a product or service. The primary problem of allowing an attacker to distribute illegal data or to cause damage to the system can effectively not be prevented with the presented solutions, however.

4.2 Excluded Problems

Solving the problem of illegal data inclusion in a decentralized database is a task that makes it likely to overlook certain properties of a complex system such as Bitcoin. To analyze the problem, grouping potential avenues for data inclusion seems sensible. This section deals with the exclusion of some ways to include illegal data. Ultimately, all excluded problems should not be considered similarly large roadblocks to a "clean" blockchain as the problems this thesis tries to solve. A short draft or explanation of potential solutions to these side issues should make this clearer to the reader.

4.2.1 Minor Data Inclusion

Bitcoin transactions and blocks offer many user-controlled sections that make the hiding of data possible. However, more or less strict rules apply to the makeup of transactions and blocks, so not every part of these data structures is equally open to abuse. Additionally, illegal scriptures are often of a certain length because they are encoded as image or video files. Finding short strings or data chunks that might be considered illegal by a certain country is difficult, and linking to their occurrence in the blockchain can be longer than redistributing the data itself. The real problem lies in the inclusion of larger files. One of the milder forms of such an attack might utilize the illegality of copyright infringement, e.g by including a music file, news article or copyrighted picture into the blockchain. Examples of more serious cases are revenge pornography which infringes on the human rights of the depicted persons, or otherwise illegal pornography whose ownership and transmission carries various risks. Protecting against these cases while excluding theoretical cases where only a few (illegal) bytes are stored on the blockchain thus seems like a reasonable problem definition, especially considering that hiding a few bits of information is generally possible in most user-controlled fields of easily reachable online services.

4.2.2 Fragmented Data Inclusion

Even larger data inclusions might not cause the blockchain to become illegal, as long as the included file is heavily fragmented because only small data inclusions were possible. The recombination of all fragments could become hard enough that it is an impossible attempt, and thus, nobody could access the data. However, one could imagine a scheme where the attacker releases a software that can scan the blockchain for multiple data inclusions that starts with a certain short byte sequence. If the byte sequence is found, all data after it is automatically recombined, making the data accessible to the user. If the scheme is robust enough, even large fragmentation might not stop it from working.

We excluded this problem for multiple reasons. For one, jurisdictions might only consider it illegal data transmission or ownership when this search byte sequence or another recombination secret is also known to Bitcoin nodes. As Bitcoin nodes are under no obligation to store such a byte sequence or secret, they would remain legal to operate as long as they do not go out of their way to retrieve such content. There are some services that operate like this to remain legal, namely cloud space providers: While they cannot prevent users from storing copyright infringing materials, they never store the decryption keys to the data, or provide them to third parties. As they technically cannot access the infringing data, they are then not held responsible for their distribution.

Additionally, the few data fields that will potentially remain viable for such fragmented data inclusion might be removed in future versions of Bitcoin. An example of such a field is the value field that denotes how many Bitcoin will be received by an output. By encoding a few bits of data in the least significant bits of the amount field, a small amount of arbitrary data can be stored within Bitcoin transactions, and not many additional bitcoins are needed. However, solutions like confidential transactions [Max18a] might solve this problem in the future.

For these reasons, we do not aim to stop small data inclusions within various user-controlled fields from occurring with our current proposal.

4.2.3 Partial Hash Collisions

Similarly, inclusion of data through partial hash collisions remains a problem. An attacker utilizing this technique to store data essentially needs to recreate hashed data structures multiple times until some of the bits resemble (part of) the data the attacker wants to include in the blockchain. As the ability to store data with this approach is quite limited, and because a fix would likely require a complete change to the utilized hashing functions of Bitcoin, which would exceed the scope of this thesis, our solution does not yet include a fix for this additional problem. However, it does not rely on any specific traits of SHA-256 or RIPEMD160, and does not conflict with a system where other hash functions are utilized to conquer this remaining problem. Additionally, this approach requires the attacker to create a more elaborate explanation on how to access the potentially illegal data, similarly to how large data inclusions require it. It is thus sensible to present a solution for the remaining, more pressing issues of data inclusion, as this will effectively reduce the toolkit of an attacker when they wish to permanently store data on the Bitcoin blockchain. On top of this, the chapter about previous work3 mentions a potential solution to this problem that was released while this thesis was being written.

4.2.4 Data Inclusions by Bitcoin Miners

Another way to include data quite freely is given to miners within the coinbase transaction, where they are free to include a few arbitrary bytes in the input. Stopping this is very well possible by changing the layout of this special, first transaction, but some of the provided space might be used to signal protocol enhancement adoption or mining pool specific information, which would need some sort of standardization so arbitrary data inclusion can be avoided. More convoluted techniques for data inclusion, such as changing the order of transactions within a block so they recombine to specific data like in the partial hash collision example are potentially preventable, too, and only present a small opportunity for file inclusion similar to those described in the fragmented data inclusion section. Ideas to solve these problems include stricter ordering rules for transactions within a block, or adding more restrictions to what or how many transactions may be added to a block. Miners are also financially disincentivized from hurting the Bitcoin network as illegal data inclusions will potentially hurt the price of their mining reward, which only becomes usable after a certain time frame. On top of that, causing the system to become illegal might destroy their investment in their highly specialized Bitcoin mining equipment. For these reasons, our solution does not yet incorporate a protection against rogue miners that want to hurt the system. It will make such attempts more expensive, however, as only very few or small miner controlled fields will remain usable for such an attack.

4.2.5 OP_RETURN

Lastly, we also do not present a solution for the only somewhat-tolerated method to add arbitrary data to transactions. It utilizes the opcode OP_RETURN within an output of a

transaction, marking it as unspendable immediately. This allows miners to stop tracking this output and thus to remove it from their memory because it will never be used in a subsequent, valid block. At the current state of the network, including OP_RETURN transactions is considered non-standard and not all mining nodes will include them into blocks they find. Furthermore, these special transactions can only add 40 bytes of arbitrary data, or they will not be relayed on the p2p network overlay of Bitcoin, making them even more unlikely to be included in the blockchain. The solution to stop this method of data inclusion is comparably easy: Simply disallow any transaction including a special OP_RETURN output. However, as long as other data inclusion methods are not completely eradicated, it would be unwise to remove this way to include data because other methods might then be utilized again, potentially causing resources to be wasted forever, such as in the case of increased RAM usage when encoding data in the receiving address of an output. In case the solution of this thesis is implemented, OP_RETURN could safely be prohibited, as the wasteful encoding of data in transaction outputs would no longer be feasible.

4.2.6 Non-standard Outputs

As a miner, it is currently possible to insert non-standard transactions into the blockchain. Some miners also seem to allow anybody to send them non-standard transactions for inclusion. Eligius is a popular example for such a mining pool [bit18a]. If non-standard transmissions are included, any kind of Bitcoin script is addable to the Bitcoin blockchain, expanding Bitcoins capabilities beyond its basic purpose of transferring ownership of potentially valuable tokens.

Retaining Bitcoin Script's capabilities while removing the possibility of storing illegal data is a potentially impossible task. For instance, one could imagine a script that requires a specific byte combination to be presented to allow the spending of the Bitcoins encumbered by the script. If the aim is to include arbitrary data, this byte combination might consist of text or image data. Thus, creating a script that calls for illegal data to spend the bitcoins protected by the script is currently a possibility, and preserving this feature in particular could be seen as going against our stated goal.

Disabling the creation of non-standard outputs is possible without functionality loss, however. In 2012, P2SH outputs [And12] were introduced. They allow a user to publish the actual Bitcoin Script only when an output is meant to be spent, similar to how P2PKH outputs do this for the public key for a transaction. Some minor limitations regarding the script size may apply, but they could be changed in the event that non-standard outputs are completely prohibited in Bitcoin. For this reason, our solution does not preserve the capability of allowing non-standard outputs to be included in future Bitcoin blocks.

4.3 Preimage Solution

If we still want to protect against illegal data being retained by the blockchain, all insertion methods for non-technically necessary data have to be removed or made unappealing. The following sections will describe how this can be done for simple transactions.

4.3.1 Signatures and ECDSA

The paper by Matzutt et al [MHH+18] describes many popular ways to insert arbitrary data into the bitcoin blockchain. All variants that store the data in standard transactions outputs can be mitigated by requiring proof that the variable parts of the standard transaction are clean. There are two types of proof that we need to consider.

The first is proof that a public key is indeed a public key (and not arbitrary data). Incidentally, the proof for this is just a valid signature that can be created with the private key that is known to the owner. ECDSA public keys are derived from their secret key, which would be cryptographically hard to find if arbitrary data was to be used as a public key. If it was not, someone could use the scheme to find the private key for some arbitrary string that happens to match the public key of another user of ECDSA, gaining their secret key in the process. Without the secret key, a valid signature for an alleged public key cannot be created, or ECDSA would be considered broken. Thus, arbitrary data cannot be stored in place of a valid public key as long as the system is expanded by the requirement for a valid signature to be provided whenever a payment to a public key address is included in a block.

4.3.2 Preimages and SHA256

The proof that an address hash as used in P2PKH and P2SH outputs is not arbitrary data behaves similarly. By providing the input data of a hash function (also known as the preimage of a hash) and the resulting hash, one can prove that the resulting hash is not arbitrarily chosen, as it would be cryptographically hard to find a preimage for a given hash. It can be assumed that his is the case, as Bitcoin would be broken in case it was possible to construct valid preimages to existing SHA256 hashes by e.g. manipulating some parts of this preimage. This could allow an attacker to exchange old blocks for completely different ones, for example, and would break Bitcoin quite drastically as it would cease to be an eternal and close to impossible to manipulate data structure. We can thus assume that SHA256 preimages cannot be created merely by knowing a resulting hash, which means the proposed solution should be secure.

As arbitrary data inclusion in Bitcoin ceases to be a problem when Bitcoin cannot be used anymore, the proving scheme may assume that an attacker cannot create arbitrary hash collisions. Knowing the preimage of an address should also not allow attackers to do anything else in regard to attacking the hash or public key validation itself. If it did, however, the scheme could be slightly changed to change Bitcoin to require a third hashing iteration so the initial hash is not known again.

The script or public key for the spending of a P2SH or P2PKH output is always known by the owner of the address, and thus the preimage is easily creatable for them. Through a curious design decision, Bitcoin even makes it easy to distribute this preimage, as the standard Bitcoin hash functions generally hashes at least once with SHA256, and the result of this first hash function call can be released as the preimage (= proof) that the second hash (included in transactions) does not consist of arbitrary, attacker-chosen data. This possibility has been briefly discussed by Bitcoin developers before and is called $P2SH^2$ [Max18b], but no implementation or elaborate explanation about its attributes exists to the best of the author's knowledge.

There is also a small caveat. A sufficiently motivated and wealthy attacker is still able to encode a few bytes of arbitrary and potentially illegal data per output. To do so, they have to create many different private keys and derive address hashes or public keys from them. By doing so, they will eventually find hashes or public keys that start with a certain bit combination. Finding a SHA-256 hash with 32 attacker chosen leading bits can be done in less than a minute on current GPUs if we only require the hash preimage and not the public key. A dictionary with such hashes and their preimage could be created and made available to attackers. Still, it would reduce the efficiency to include data in outputs drastically. Whereas an attacker can add 256 bits now, this amount would be reduced by 87.5 percent. With more computationally intensive hash functions for Bitcoin addresses or by elongating the proof and address scheme with additional data as a salt, this attack vector can be further mitigated.

Proofs are not required to be stored once many blocks have been created on top of older transactions, and can thus be discarded. This makes it unlikely that someone will store illegal data within the proofs, as they won't gain the benefits of storing data eternally if it won't be part of the blockchain.

As an example, proofs for transactions within an old block might be required until 100 blocks have been built on top of it, or else a block will be deemed invalid and discarded. After 100 blocks, nodes will not request or require proofs for the older block. Then, after 200 blocks, the nodes storing the proofs can safely discard them as a competing blockchain is unlikely to cause a block reorganization that spans over 100 blocks. 100 blocks is a special number in Bitcoin as miners can use their block creation reward after these many blocks are created on top of one of their mined blocks. A reversal or attack that removes over 100 blocks with a competing chain with more blocks would be very dangerous for Bitcoin, as most miners would then lose a lot of revenue, potentially causing them to stop mining Bitcoin and reducing overall security.

Abuse of the proof system is unlikely and much less severe than an abuse of Bitcoin for illegal data storage. An attacker storing data within proofs would only profit from this for less than 36 hours during normal network operation (e.g. less than 200 blocks), as that's the time frame within which his proofs would be stored and transmitted by the network. The legal system often works much slower than this, so it is unlikely that a worldwide network could be in agreement over the legality of data that is only stored fleetingly, anyway. Additionally, an attacker would lose the bitcoins sent to the hashes of these preimages permanently, as the preimage is data, and not actually a hash of a valid public key that would allow them to spend the coins again. All this makes the attack comparatively expensive, and unlikely to have the desired effect.

If this proof system does not prove sufficient for any reason, another option would be an implementation of the scheme described in the paper by Matzutt et al [MHZ+18] which was released during the time this thesis and the accompanying code were being written. They present a special script for outputs that incorporates a more elaborate proof that no arbitrary data was included, with the downside of an increase in transaction size. The following parts of the solution do not require anything of the proof system than stopping an attacker from including data, however, so their solution would fit our purpose, too.

4.4 The Solution for more Convoluted Transactions

The remaining, larger attack vectors all use inputs to store data. P2PKH and P2PK outputs require valid signatures or valid signatures and the corresponding public keys to spend them. P2SH outputs, however, are more flexible in what they require to be spent successfully. This is a problem, as this flexibility allows for data to be included in a much more straightforward way than hiding it within fields meant for public keys or hashes.

One example for an efficient way to store bad data using P2SH inputs is the following: Create a P2SH output that consists of a hash for a script that has a branching instruction. The non-taken branch pushes arbitrary data onto the blockchain, whereas the branch to be taken does (for example) a standard check for a certain public key plus signature. We then spend the P2SH output by creating a new transaction which spends the output by providing this script and the needed public key and signature. As the transaction is valid, it should be included into a block without problem, even though it has a non-standard structure. Requiring some sort standard structure after a P2SH has been made would even be problematic, because the person being paid cannot change the script without changing the hash of the P2SH. If the original script were not to be accepted because of rule changes in Bitcoin, he would basically lose the Bitcoin he has received via the

P2SH output. Considering the rules being followed in development of Bitcoin, this would probably be an unacceptable change for owners and developers alike.

This remaining way to store data on the blockchain is not as problematic as storing data in outputs, however. Whereas unspent outputs reside in RAM on every Bitcoin node, and need to do so for block validation purposes as it prevents double spends, spent outputs (and their inputs) only reside on the hard drive of most Bitcoin nodes. They can even be discarded by a portion of the network by pruning, which effectively deletes the unneeded data once it has been scanned to build up the unspent outputs database in RAM. Such nodes are obviously also not in danger of transmitting illegal data to other participants. They might however request the data initially, storing it for a short time.

A change to how Bitcoin P2SH inputs operate and what Bitcoin guarantees for P2SH could present a solution to the remaining data inclusion problem. First, the effects of a potential straightforward removal will be explained and discouraged. Then, the option of how a safe solution can work will be discussed.

4.4.1 Plain Removal Impossibility

If we assume a permanently deleted transaction within a Bitcoin block, Bitcoin participants could still verify the proof-of-work of the block. What fails, however, is the block validation. One important possibility to consider is that the transaction whose data cannot be acquired anymore and that might be referred to by later transactions could have always been invalid. It is also not possible to know how many Bitcoins later transactions are allowed to spend when they include an (equally deleted) output from the deleted transaction. Even worse, the whole block reward the miner got for including the transaction cannot be verified anymore. If miners had worked together at any point in the past, it would have enabled them to acquire free bitcoins, and to create more bitcoins than should be possible. Owners of the bitcoin being spent within the deleted transaction would also immensely benefit from such a scheme: Without any proof that they ever used their bitcoin, they could use the same outputs again in a new transaction, effectively causing a double spend and a doubling of all involved bitcoin at the same time.

4.4.2 Fine Grained Removal

Now that it is clear why removing a complete transaction is not feasible, the downsides of more fine-grained deletion of transaction data can be explored. Not every section of a transaction lends itself for data inclusion, and the output portion has already been hardened by the previously described solution.

The section that remains abusable is the ScriptSig section of every input of a transaction. This is illustrated by an excerpt of the developer documentation for Bitcoin. The redeem script hash can be provably clean of illegal data through the scheme described previously. The (optional) signature cannot contain arbitrary data, either, or it would be invalid, stopping the inclusion in valid Bitcoin blocks. Thus, only the full redeem script causes data inclusion to remain feasible. Essentially, allowing removal of this part of an input would be enough to harden the blockchain against remaining effective ways to include potentially illegal data. However, this part of the input contains the proof that a P2SH output has been successfully spent by the owner, so allowing the removal of this proof is not easily possible. Another smaller problem is that inputs are also hashed and committed to by signatures, so removing some data of an input would invalidate the signatures of the transaction. This, again, would render the whole transaction invalid, potentially forgeable and a proper transfer of ownership of the coins would become impossible, similar to the case where a whole transaction is deleted.

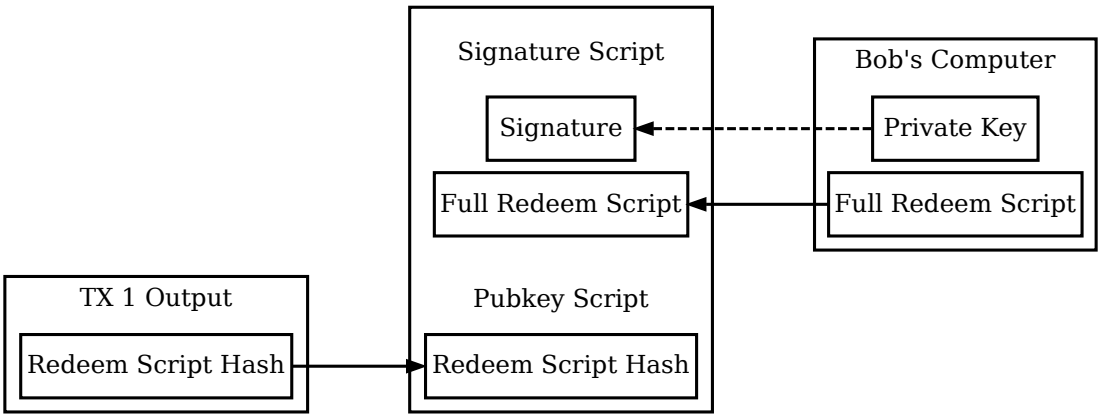

Figure 4.1: Whereas the redeem script hash cannot be abused for data inclusion anymore, the Full Redeem Script might still contain illegal data. P2SH censorship solves this problem without touching other transaction data. This graphic was taken from the Bitcoin Developer Reference [bit18c]

Our proposed scheme can solve this, however. Instead of storing the input data directly, the Bitcoin blockchain could store a hash of the input within the basic transaction data structure. Then, the hash is used as a reference to the real input stored elsewhere, potentially after the end of a normal Bitcoin block, in a data structure outside of the blockchain that is designed to allow deletion. This does not create similar problems as deletion or modification of data within blocks for the following reasons: If the input data is replaced by a hash, the signatures that sign a transaction can include these hashes instead of the input data when it is created, retaining the goal of making the transaction data unchangeable after transaction publishing or block inclusion. The input data cannot be changed as this would break the hash, but now deletion or just not transmitting the data to other participants is possible. Other hashes used in Bitcoin, such as the transaction id or the merkle root hash will not be broken in this event, as the signature will not sign the deletable data directly. This change to the data structure is a requirement for the next part of the solution where we explain how deletion will be done without making any particular group in the Bitcoin ecosystem too powerful. Figure 4.2 shows and explains the changes that have to be done to the layout of a transaction with a single P2SH input.

4.4.3 Pay for Removal

The remaining problem is the following: Deleting the inputs removes the possibility of confirming that a transaction was actually valid by other nodes or more specifically by nodes that joined the network at a later time. These notes will never see the complete transaction, so they might assume that the transaction was never valid in the first place, and the removal function was used to hide this fact. To counter this, our solution proposes the following: Should a transactions input be available, its data is used to validate the transaction just as before. The system thus works as before.

In case an input (or more specifically, the ScriptSig part of an input) was deleted or is simply not transferred by other nodes, however, a later block has to include a special removal transaction that destroys the same or larger amount of bitcoins to make up for the removed input. These special transactions that mark data as deletable should only be

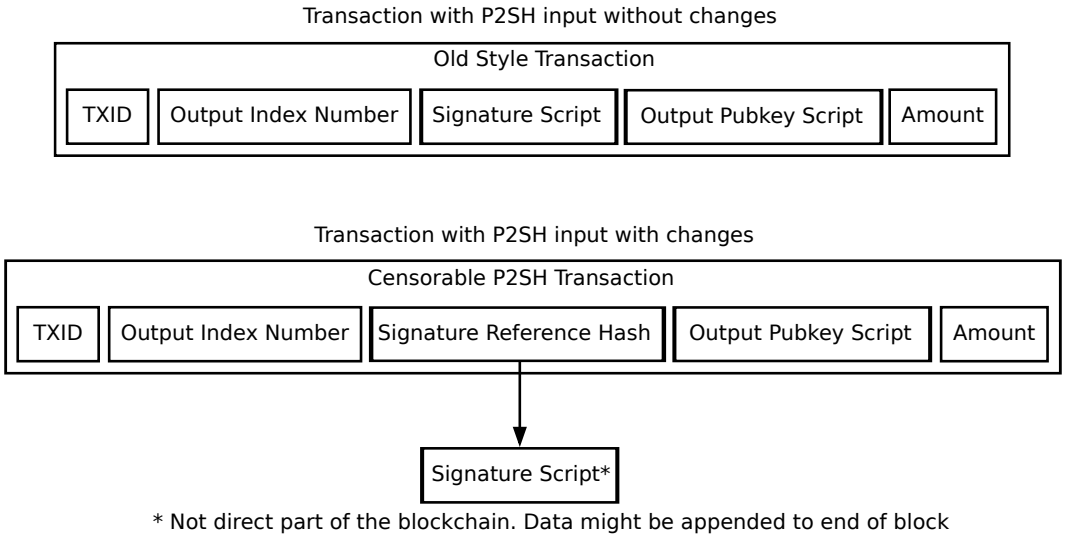

Figure 4.2: Changes to the transaction layout for P2SH transactions. Instead of storing the Signature Script in the transaction itself, only a referencing hash is stored. The real Signature Script will be kept in a datas tructure besides the block. Options include the creation of a merkle tree that then commits to all P2SH reference hashes, and that might be included in the coinbase transaction of a miner to secure integrity of sent data. A deleted Signature Script is not a problem as long as a future block references the same hash and provides sufficient funds. This figure is an edit of a graphic from the Bitcoin Developer Guide [bit18c]

able to use bitcoins held in P2PK or P2PKH, or other provably clean outputs, as our first solution stops them from containing bad data. Without this requirement, special removal transactions might perpetuate the problem of illegal data inclusions on the blockchain, so we propose to prevent this by adding this requirement. This does not restrict Bitcoin users from creating censorship transactions in any meaningful way, as moving bitcoin into an output that may pay for a censorship transaction is always possible with an additional transaction. A layout draft for this type of transaction can be seen in figure 4.3.

4.4.3.1 Averted Attack Vectors by Requiring Payment

This unintuitive requirement of spending an equal amount of bitcoins to allow the deletion of an old input is important for two reasons: If it was not required, miners could try the following attack scheme to enrich themselves: After secretly mining an incorrect block with a transaction that spends outputs that they do not control, i.e. that they do not know the required signature for, they create further blocks that ask for the deletion of those inputs in a later block. Similarly to the first block, this would be done in secret. After the blocks are created, they are delivered to other miners all at once. If the miners instigating this attack find all blocks fast enough, this would allow them to spend arbitrary bitcoins stored in P2SH outputs. As finding valid Bitcoin blocks is essentially based on luck, even a miner controlling only a small subset of all available Bitcoin hash rate might be able to pull off this attack if he is lucky enough. This is different from a so called 51 percent attack because the potential reward is much larger than the block reward, with the limit being the number of bitcoins being stored in P2SH outputs, or the maximum block size of a Bitcoin block. An attack like this could be hard to detect if the number of blocks between transaction inclusion and removal transaction is kept low. If it is larger, for example 100 blocks, then the attack is likely to be detected as many blocks of the normal

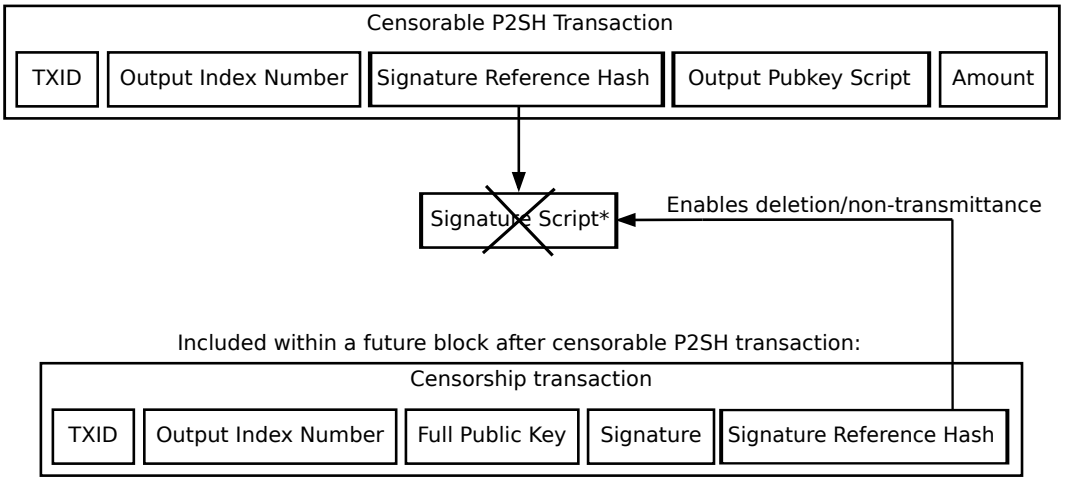

Figure 4.3: Instead of an output and an amount, a censorship transaction must only reference the Signature Reference Hash of a previous transaction. The referenced input data of this transaction, also called signature, can then be deleted and does not need to be transmitted to other nodes anymore. In case of a full blockchain revalidation, these transactions should be scanned for right after blocks have been checked for a valid proof of work, so the block validation process is aware of censorship transactions when it encounters blocks without data. This figure is an edit of a graphic from the Bitcoin Developer Guide [bit18c].

Bitcoin network would be orphaned in the process. This would cause further problems that are generally unlikely to occur with Bitcoin as it is now, but if no payment was required, the potential return on investment of such an attack might still incentivize these types of attacks that were previously disincentivized by the extreme costs. Another variant of the attack might try to target Bitcoin miners or internet infrastructure for multiple hours to pull off this heist without it being similarly apparent. As Bitcoin miners are distributed all over the world, a broken internet infrastructure would have miners expect their blocks to become orphaned.

All this is solved by requiring an equal amount of bitcoins to be used to censor. Censoring faked transactions essentially becomes a zero sum game for the miner, as they would have to pay the same amount of bitcoin as they would be able to steal with the scheme.

The second reason for requiring a payment is based on the problem of what to do with the P2SH outputs that were used in the censored transaction.

If we were to consider them spent, the creation and deletion of an actually invalid transaction by a miner as described above would allow the miner to destroy the wealth of the owner of the P2SH output. This, again, enables miners to do something that they were not capable of before and is probably not in the best interest of the users of Bitcoin.

If we do not consider such outputs as spent anymore but made deletion free/cost less, then the owner of the P2SH outputs could request the deletion themselves. This might allow them to double their wealth, because they could then use the outputs of the new, censored transaction, while the old outputs would become spendable again. Besides users, miner could also create such transactions, paying themselves huge fees, only to get their money back by censoring their own transactions.

The only viable solution seems to be to require an equal or higher amount of bitcoin to censor a transaction. In this case, a miner or user censoring their own transactions

does not cause them to gain more bitcoins in the process. At the same time, the censored transaction's outputs become available again, meaning a single miner cannot cause a person to lose their funds by starting an attack similar to the one described above.

4.4.3.2 Caveat

A caveat remains, however. If someone else were to pay for the deletion of the input, the user originally creating the transaction can potentially become more wealthy, just as in the scenario where deletion is free.

To do so, they would create a transaction containing illegal data within the input of a P2SH input. The outputs of this transaction would pay himself or pay for something they are interested in. After calling attention to the illegal data and miners or other users voluntarily paying for its removal, the outputs that were spent by the illegal data inputs would become unspent again. The outputs of the censored transaction would remain valid, too, essentially doubling the money of the attacker. This could incentivize people to add bad data to the blockchain, in the hope that other parties that need the blockchain to remain legal pay for the removal of the data.

While bad, it can be argued that this is a more easily solvable problem than the allowance of illegal data on the blockchain. For one, illegal data might make Bitcoin permanently illegal in most countries. Compared to that, the monetary damage caused by attackers creating transactions that essentially pay them some money if he is successful is smaller. Furthermore, it seems possible to create protection mechanisms against the user enriching scheme described above: For instance, if the data was actually shown to be illegal, miners could just opt to not allow spending of the initial UTXOs into any future blocks. If another miner was to include them anyway, they might be held accountable in their respective country, requiring them to pay for the subsequent removal similar to how laws call for damages to be covered by those that caused the damages.

This approach could be strengthened by combining it with a voting mechanism based on recently created blocks that requires miners to use a portion of their block reward income for re-censoring a new transaction that is spending outputs that had already been used in a previously censored transaction. By only allowing block finders to vote, sybil attacks can be averted. Additionally, as not all miners are completely anonymous, active miners voting for re-censorship despite there not being any evidence for criminal behavior in the first or second censored transaction could possibly be held legally responsible for misusing this miner accountability mechanism. The security of such a mechanism would probably rely on the honesty of the majority of the miners, which isn't perfect for a system that empowers miners to hurt their competitors. In an environment where the majority of miners have to work inside the laws of various countries, this capability might not be abused easily, however. It is also worth noting that a solution to the illegal data inclusion problem might not be necessary in the first place if miners operate outside the confines of various legal systems, so maybe it is not a problem to assume that major parts of the network may be fineable by governments in case they abuse their powers to unfairly hurt their mining competition. Ultimately, a potential for a limited mining revenue loss might not be as much of a problem as the illegal distribution of prohibited data, so the downsides should be weighed against the potential for someone to gain money by having their transactions censored.

4.4.3.3 Precise Censorship

Another important detail about this technique is its potential for more precision when illegal data is to be censored. If we consider that in most cases, not every part of a file is equally incriminating, and that the inclusion via P2SH inputs requires the use of many

inputs which can be individually censored, the overall impact of the mechanism can be reduced because it won't be necessary to censor all inputs of an offending transaction. If not all inputs need to be censored because of input size limitations, the creator of the transaction has to pay most of the transaction fees themselves, further reducing the profitability of a scheme that involves including bad data in the blockchain.

Another upside of not censoring the complete data is attesting the previous existence of incriminating data to other miners and Bitcoin users. For instance, libelous texts or pornographic images might still be somewhat recognizable even if they are heavily obscured by deleting their most incriminating parts. This means any Bitcoin user can check partly censored transactions for their remaining content and make up their own mind if the transaction creator intended to abuse the system. Incidentally, the mere fact that some inputs can be interpreted as data should serve as a good indicator that the transaction creator had non-standard intentions with their transaction. In some cases, it might even be possible to tell that the deleted portion of the data was likely illegal, allowing miners to verify this fact for themselves.

Miners who are sure to have been part of the correct Bitcoin network at all times may also opt to disable outputs used in censored transactions in every case. The non-existence of orphaned blocks allows them to be sure that the owner of the coins was able to spend his coins once. Similarly, the miner's node probably still owns the proof that was marked for deletion, and it knows that it once validated the proof, so it could not have been an invalid, miner created fake transaction that has been censored later to appear genuine. In effect, there are only two cases when a miner should feel compelled to enable outputs used in censored transaction to be spendable again: In the first case, the miner re-enables outputs because he assumes that a large premining attack has been taking place during the time he operated, as this might have been an attempt to make coins unavailable to their owner by using them in a faked and later censored transaction. The second case is a new miner that joins the Bitcoin network without knowing any prior history besides the blockchain. For outputs used within transactions that cannot be trusted to have been censored legitimately because all proof for illegal data inclusion was removed, these new miners should ideally not assume that the censorship was legitimate for the same reason as above. However, if sufficient proof can be made available to new miners, they should obviously opt to never include such a transaction again, or risk being held responsible for the monetary loss of later illegal inclusion attempts.

4.4.3.4 Retaining Eternality

The proposed solution does not distinguish between different types of P2SH outputs. This means that even benign P2SH outputs that could be provably free of bad data can be censored. This is actually not needed and could be prevented. Further updates to the proposed scheme could allow proofs similar to the ones discussed for P2PKH and P2PK transactions. If a P2SH output (or its corresponding future input) could be marked as requiring such proofs for inclusion, censorship resistance could be reintroduced for just those P2SH outputs that are provably clean of bad data, as long as the parties involved also actually provide said proof. Only P2SH outputs that require data that is not provably clean, such as elaborate scripts designed for a single, special occurrence, or scripts that require data to be included in the input would then remain censorable. This could re-enable some yet unknown use cases that actually rely on the censorship resistance of all transaction data for P2SH transactions, at least as long as those use cases allow for proof creation.

4.4.3.5 Miner Acceptance

What remains is a system where it is much more inefficient to add off-topic data to the blockchain. At the same time, it is impossible to have a benign user lose their funds even if a miner implemented the above attack, while also stopping miners and users from enriching themselves by conjuring censorship transactions from third parties. The costs of this approach are the possibility of minor censorship of only the input data if P2SH is used, whereas the rest of the transaction stays intact. Additionally, miners might be held responsible for repeat inclusion of bad data, or if they mine blocks that contain transactions that spend outputs that were previously part of a censored transaction. This essentially expands the requirements for miners to non-technical areas, which can be seen as a downside of the technique. At the same time, however, miners are also part of the victims in case Bitcoin or its blockchain were to be illegalized because of data inclusions, so it is not completely unreasonable to require them to be more cautious about transaction inclusion in these special cases. Furthermore, just not mining previously censored transactions could be a viable strategy for miners that cannot do these checks, allowing other, more sophisticated miners to specialize on safe re-inclusion if the owners of the coins provide proper evidence such as the potentially clean, but removed part of the transaction.

4.4.4 P2SH Idea Implementation Details

Not every idea is appliable in real software. This section deals with the complexity of changing Bitcoin transactions in a way that allow for the fine-grained removal of the script signature part of a transaction.

4.4.4.1 Changes to Data Structures

Implementing this scheme requires reconstruction of basic Bitcoin data structures. Transactions, or more specifically their inputs must not be stored directly inside a transaction anymore, but should only be referenced with a hash. If an input turns out to be problematic, its data can be removed. The transaction itself, on the other hand, will remain intact and its id will not change.

Similarly, an input also consists of multiple fields, and only one of these fields can be used to effectively store data. If this part of the input were to be replaced with another layer of indirection, i.e. a hash that only references the input's ScriptSig, censoring included data of a transaction would essentially mean just deleting the ScriptSig that is stored outside of the input. Other important data such as the id of the referenced previous transaction, the index of the unspent output, the script length and the sequence number would not be touched and would thus remain useable and trustable to, for instance, determine what output has been used to include the censored data.

4.4.4.2 Network Protocol Additions

Another important implementation detail is how the network should be notified about the event of censoring a single or more inputs. One way to implement it could utilize SegWit transactions with a higher version number or a length unlike the current standard SegWit transactions. A straightforward approach would be to push the transaction ID, followed by a list of input indexes that are to be censored. When validating this transaction, the existence of the transaction in a prior block would be confirmed. Additionally, the output would be required to contain an amount of bitcoin equal or greater to what the input provided previously. While not required, it might be sensible to restrict inputs in a censorship-transaction from being non-standard. No inputs of a censorship transaction could then be deleted and thus require the knowledge of the data (and validity) of a later

censorship transaction that censors the original censorship transactions. This simplifies the checking of censorship transactions and does not enable an attacker to cause a huge lookup overhead for Bitcoin node operators as it stops the need for recursive censorship transaction checking.

4.4.4.3 Performance Impact

Performance wise, the scheme should not cause block verification to become a problem if attacks occur rarely. First of all, censorship transactions should only have a meaningful effect on performance in case transaction input data is actually not procurable. In case of censorship requests for benign inputs, Bitcoin can behave just like before, incurring no performance penalty beyond that of a other transaction types.

Additional checks are only needed if the input is actually not procurable by any peers of the network, or in case the operator refrains from requesting it for legal reasons, for instance if they were provided a list of bad inputs by a government or similar entity. In this case, verification of the censored transaction requires knowledge about its later occurring censorship request transaction. Finding later occurring censorship requests will require additional in-memory data structures to be held current, similar to how unspent transaction outputs are kept in RAM. After confirming the output of a censorship request transaction is sufficiently large to pay for the censored input(s), it regards the censored transaction as sufficiently validated. Block validation continues as before.

It does incur another performance penalty if locating censorship request transactions is not supported by e.g. annotating block data in some way, such as adding these special transaction to a specific portion of a block that will be easy to parse directly after the initial block download or before the first occurrence of a censored transaction. The specific implementation details of this are not very important, however. If we take into consideration that censorship only complicates the initial validation for new Bitcoin nodes, even a bad implementation scanning every byte of every block for such transactions should not cause performance to drop meaningfully for nodes that are already synced to the current network and receiving blocks as they are created. Similarly, when nodes request only a few hundred new blocks every few days, such as in case that a node was to be offline or under maintenance, additional performance costs will be negligible. Parsing the blockchain data is generally not considered to be a bottleneck compared to the resource consumption of validating ECDSA signatures.

4.4.4.4 No Impact for Non-Censoring Nodes

Even better, nodes that are continuously mining could essentially completely ignore censorship request transactions as long as they do not want to censor P2SH transactions. It is only when a node operator decides to agree to the request of deleting and not transmitting certain inputs anymore that they need to check that the amount in the special output of the censorship request transaction is sufficient to pay for the censored inputs. In case they did not, new nodes joining the network would not receive the censored inputs, and find no valid censorship transaction that essentially protects the network from being exploited for financial gain via fake censorship. New nodes would thus not consider the currently longest PoW Bitcoin blockchain valid despite it having been valid in the past, and a minority of the hashing power would fork into a separate blockchain. It is thus also in the interest of every node operator to check the validity of censorship request transactions, but one could also imagine that only nodes that do censor check the validity of censorship requests, and if they are insufficient, the transaction is just ignored until a sufficient request comes along.

5. Proof of Concept

To demonstrate the concept of proving the non-inclusion of arbitrary data, a proof-system for hashes and public keys has been implemented to serve as a proof-of-concept. To increase the credibility of the proof of concept, it is implemented as a change to the most popular Bitcoin node implementation, namely Bitcoin Core in its recently released version 0.16. As the original code is written in C++, the code consists of minor changes to existing files and a few newly created ones. These new files contain the most important additions, enabling a fast familiarization with the proposed changes. In total, around 800 lines of code had to be written. Modification of existing code lines was not necessary, but some existing functions had to be extended with additional lines of code.

5.1 Bitcoin Core

The layout of Bitcoin Core is not easily understood at first. While most files do serve a specific purpose and seem to have a high cohesion, the general lack of comments in key parts of the source code requires a considerable time investment to be able to find the parts that might be relevant to consider when adding new functionality. While developer documentation does exist, it is mostly aimed at explaining the general idea of Bitcoin, and how it is meant to operate. Bitcoin Core, however, is the reference implementation, and in cases where the documentation is unclear or faulty, the behavior of Bitcoin Core would be generally considered correct. This means that someone unfamiliar with the Bitcoin Core codebase has no other choice but to investigate the concrete implementation details within Bitcoin Core if no sufficiently elaborate documentation is found.

Writing this proof of concept within Bitcoin Core was only possible through searching the code base by hand, trying to find the functions involved in transaction validation, hash creation, and signature verification. At first, no IDE was used, but the debugging functionality of Eclipse proved invaluable at later parts of the implementation, as using the debugger by hand did not provide the same overview of the unknown codebase.

After all important source files had been discovered, implementation of the proof of concept was mostly straight forward. Debugging before the full functionality had been implemented was made more difficult by some security functions of Bitcoin Core, namely the validation rechecking of old blocks on startup. This was solved by the frequent re-generation of the testing environment, which was made easy by a tool that is presented in the following section.

5.2 Proof of Concept Usage

Bitcoin consists of a p2p network of computers running Bitcoin node software. Some of these nodes create new blocks and solve the proof-of-work target to get their block appended to the current blockchain, others merely relay transactions and check the validity of the blockchain and new blocks.

Testing new features and trying out recently implemented functions on the real Bitcoin network is complicated, as transactions might require a fee to be transmitted, resulting in monetary costs for testing. For this reason, testing is normally performed using an official test network, where test bitcoins are somewhat freely handed out. This blockchain is completely different from the normal one, and distinct markers make sure that there is no chance for confusion in regard to real and test bitcoins when using the software, addresses or transactions. However, it can only be used to test new features that have been properly proposed and included in a recent beta release of the software, as participants of this test network still need to install new software versions manually. To solve this issue, Bitcoin Core has a feature called ”regtest”. It basically allows for the creation of a miniature Bitcoin network by simplifying block creation and allowing a programmer to spin up their own, tiny Bitcoin test network running on a single computer. On top of this, Sean Lavine created a preconfigured environment[[Lav18]] that interconnects two Bitcoin Core instances and allows for a more convenient usage of the regtest functions. As this was ideally suited our the proof of concept, the project's Bitcoin core configuration files are used to simplify the deployment of a usable, localized test network where the implemented code changes presented within this thesis can tried out. However, someone already familiar with setting up a testing environment for Bitcoin does not need to use these simplifications, they are merely an additional convenience.

The following steps will create a test environment that allows for the testing of the proof of concept code.

5.2.1 Test Environment Setup

At first, Bitcoin Core needs to be checked out and built. These steps assume a linux environment, and the author recommends running these steps within a Linux VM (e.g. Ubuntu, Fedora, etc.) in case another operating system is used as it seems to be the most popular environment for Bitcoin development.

Listing 5.1: Git checkout of Bitcoin Core code

```
mkdir proofofconcept
cd proofofconcept
git clone git@gitlab2.informatik.uni-wuerzburg.de:s318196/bitcoin.git
cd bitcoin
./autogen.sh
./configure
make
```

In case of problems or unfulfilled requirements that autogen cannot handle, it is recommended to read the instructions in the ”doc” directory. The proof of concept code does not introduce any additional requirements to the building environment, meaning that an environment that can build the official Bitcoin Core software will also be able to build our modified version.

The make command should have created multiple files in the ”src” directory. Among them are ”bitcoind”. Check that ”bitcoind”, ”bitcoin-cli” and ”bitcoin-tx” can be found and executed within this directory.

As our next step, we will create the test environment using the aforementioned bitcoin testnet box environment.

Listing 5.2: Setup of Bitcoin Testnet Box

```
cd ..
git clone https://github.com/freewil/bitcoin-testnet-box
cd bitcoin-testnet-box
make start # starts both bitcoind instances
../bitcoin/src/bitcoin-cli -datadir=1 generate 101 # creates enough ↙
    ↳ blocks to have spendable test bitcoins
```

This setup is sufficient to run all tests presented in this thesis. There are two additional commands that might be useful in case the code is rebuilt for debugging purposes or to test further changes:

Listing 5.3: Bitcoin Testnet Box commands

```
make stop # stops the environment to start it again with freshly b↙
    ↳ uilt binaries
make clean # deletes the blockchain. It then needs to be generated a↙
    ↳ gain or not enough test bitcoins will be available
```

This concludes the setup part. What follows is a short explanation of the added command-line-interface commands, and their purpose. After their introduction, a thorough test-case with in-detail explanations of all important aspects is presented.

5.2.2 Commands

Bitcoin Core has multiple RPC commands that can also be triggered via a command-line-interface. The implementation of the proof of concept for the proof system basically consists of some hooks inside functions that are called when transactions are verified. The RPC commands were created to make new proofs and to import them, and to generally activate the new rules. This approach was chosen because a complete overhaul of the transaction system, i.e. creating and including the proof into transactions, would have required multiple patches to various parts of the system. While doable, the time restrictions on this thesis did not allow it, as implementation errors would most likely have been very hard to track down. Additionally, limiting the proof of concept to transaction validation functions allows people interested in this proof to gain a much quicker overview over the central code changes, without confusing them with mere user-interface changes.

createoutputpkproof <address> Creates a string that can later be imported into a Bitcoin client to serve as proof that a given public key is indeed a public key and not arbitrary data. Internally, the wallet portion of the node searches for the private key of the address, signs the hash of the public key with it, and outputs this signature encoded as base58. This implementation serves as an example how the proof could be created by a user to give it on alongside their address, so other people can include it in a transaction.

createoutputpkhproof <transaction id> <output index> Creates a string that can be imported to serve as proof that a given public key hash does not contain arbitrary data. This command features an address lookup, meaning the actual transaction output for which the proof is needed can be supplied, and the corresponding address is found automatically. The preimage of the address hash is then created by finding the corresponding public key and hashing it. This implementation shows how the proof creation process for change- and self-controlled addresses can be automated internally, making it completely transparent to the user.

addknownpreimage <preimage> <address> This command imports the preimage for a certain address hash. The preimage is taken, hashed again, and stored within the client so that future transactions that use the same address can be accepted. It is important to note that this command is meant for testing the proof of concept. In a real implementation, the preimage should be supplied with a otherwise valid transaction to mitigate potential denial of service attacks.

addknownpreimagepk <signature> <publickey> Imports the proof of no arbitrary data inclusion for a public key. The pubilc key is hashed, then the validity of the signature is checked by using the public key and comparing if the signature signed the same hash. This is a function to test the proof of concept, the same caveats as with addknownpreimage apply.

enableproofrequired Enables the requirement that new blocks will only be accepted if every output is provably free of arbitrary data.

disableproofrequired Disables the block validation rules this proof of concept change to Bitcoin Core introduced.

5.2.3 P2PKH Example

The first example covers P2PKH transactions, i.e. transactions that contain just a hash of a public key. We first need to create a new transaction, then create the proof for that transaction's outputs, import them, and generate a new block that will then be accepted.

Creation of a new standard transaction can be done with two simple commands. If you haven't done so already, create enough blocks to have some bitcoins we can spend.

Listing 5.4: Create enough blocks to have spendable bitcoins

```
../bitcoin/src/bitcoin-cli -datadir=1 generate 101
```

Once enough blocks have been generated, we use sendfrom to create a new transaction.

Listing 5.5: Create a standard transaction with two P2PKH outputs

```
../bitcoin/src/bitcoin-cli -datadir=1 sendfrom "" ↙
    ↳ mpSEu9gQiQj7rws96v8cHDfu28Bj8BmGEk 1

result: 256d452843c0f9960640003ea8984c2ef00ba0a19a553c0f28c21fbb5f06↙
    ↳ ba3b #txid
```

By supplying the transaction id and output index, createoutputpkhproof can create the required proof

Listing 5.6: Create proofs for the transaction

```
../bitcoin/src/bitcoin-cli -datadir=1 createoutputpkhproof a47b99e91↙
    ↳ a866b6a7aff61959cb8a5ac1b15026cfd496b580df0c7a12ee2273d 0

result: {
  "proofhash:": "B8GxQqPnD6heAMjhkH5oexy1jZFjniti7HwQ5woLGx29"
}
```

Another proof is required for the change output address.

Listing 5.7: Create proofs for the transaction (2)

```
../bitcoin/src/bitcoin-cli -datadir=1 createoutputpkhproof a47b99e91↙
    ↳ a866b6a7aff61959cb8a5ac1b15026cfd496b580df0c7a12ee2273d 1

result: {
  "proofhash:": "Huc6VAbbmPm8786jaFfZ1emwEncrtXAVkFq4g3CVSbh7"
}
```

Both proofs can then be imported in the second and first node

Listing 5.8: Import proofs

```
../bitcoin/src/bitcoin-cli -datadir=1 addknownpreimage ↙
    ↳ B8GxQqPnD6heAMjhkH5oexy1jZFjniti7HwQ5woLGx29
../bitcoin/src/bitcoin-cli -datadir=1 addknownpreimage ↙
    ↳ Huc6VAbbmPm8786jaFfZ1emwEncrtXAVkFq4g3CVSbh7
../bitcoin/src/bitcoin-cli -datadir=2 addknownpreimage ↙
    ↳ B8GxQqPnD6heAMjhkH5oexy1jZFjniti7HwQ5woLGx29
../bitcoin/src/bitcoin-cli -datadir=2 addknownpreimage ↙
    ↳ Huc6VAbbmPm8786jaFfZ1emwEncrtXAVkFq4g3CVSbh7

{
  "Notice": "Successfully␣added␣p2pkh␣proof"
}
```

To test the new consensus rule, it has to be activated.

Listing 5.9: Create a new block

```
../bitcoin/src/bitcoin-cli -datadir=1 enableproofrequired 1
```

Creating a block will automatically include all transactions we created. If we did not add the proof, this block would be rejected. As we did, however, the block will be accepted without further problems.

Listing 5.10: Create a new block

```
../bitcoin/src/bitcoin-cli -datadir=1 generate 1
```

5.2.4 Example of a Rejected Block

We create a standard transaction sending bitcoin to the same address again.

Listing 5.11: Create a standard transaction with two P2PKH outputs

```
../bitcoin/src/bitcoin-cli -datadir=1 sendfrom "" ↙
    ↳ mpSEu9gQiQj7rws96v8cHDfu28Bj8BmGEk 1

result: fa187e21546c0e0de8ce528d466df04e8e5a7a7b9c05f64402c4d3271010↙
    ↳ ac73 #txid
```

We create another block. It will be rejected because the change address proof is not known.

Listing 5.12: Create a new block

```
../bitcoin/src/bitcoin-cli -datadir=1 generate 1

result: error code: -1
error message:
CreateNewBlock: TestBlockValidity failed: no known address ↙
    ↳ preimage, Transaction check failed (tx hash fa187e21546c0e0de8c↙
    ↳ e528d466df04e8e5a7a7b9c05f64402c4d3271010ac73)  (code 16)
```

The block is then discarded, just as the transaction would be discarded by other nodes.

5.2.5 P2PK Example

For our second example, we will manually create a P2PK transaction, create the proof, and send it to another node for block inclusion. The manual creation is necessary because the Bitcoin software does not support the creation of P2PK transactions via other ways.

To begin, we list all unspent outputs and fill environment variables with one of the resulting outputs.

Listing 5.13: List unspent outputs

```
../bitcoin/src/bitcoin-cli -datadir=1 listunspent

UTXO_TXID=e7d12b93e03975f6734dffec79c9973fea7d3a7bbb33da16d9c88de20 ↙
    ↳ 89ca2ed # enter one of the listed outputs
UTXO_VOUT=0  # enter the index of an unspent output
NEW_ADDRESS= # enter a valid bitcoin address
```

Using the following code, we create a standard P2PKH transaction that we can later adapt to our needs.

Listing 5.14: Create a standard transaction using the UTXO

```
bitcoin-cli -regtest createrawtransaction '''
    [
      {
        "txid": "'$UTXO_TXID'",
        "vout": '$UTXO_VOUT'
      }
    ]
    ''' '''
    {
      "'$NEW_ADDRESS'": 10.000
    }'''

Example:
../bitcoin/src/bitcoin-cli --datadir=1 createrawtransaction ↙
    ↳ '[{"txid":"23d653d24864a27e576ed4a5c2e9f5b43ed4f9b4fc20c33bd7d␣↙
    ↳ 84d5bd9eaedfa","vout":0}]' ↙
    ↳ '{"mt7rZGeLH1QBDaeDiULAmTJfqdc5Exh11A":10}'

Result: 0200000001faedead95b4dd8d73bc320fcb4f9d43eb4f5e9c2a5d46e577e↙
    ↳ a26448d253d62300000000000ffffffff0100ca9a3b000000001976a9148a3b4↙
    ↳ 74d38009afd977bf88be033be9f9f9c416f88ac00000000
```

Removing the first output (0) reduces the standard transaction to a shell for our P2PK transaction.

Listing 5.15: Remove output 0

```
../bitcoin/src/bitcoin-tx -regtest 0200000001faedead95b4dd8d73bc320f↙
    ↳ cb4f9d43eb4f5e9c2a5d46e577ea26448d253d62300000000000ffffffff0100↙
    ↳ ca9a3b000000001976a9148a3b474d38009afd977bf88be033be9f9f9c416f8↙
    ↳ 8ac00000000 delout=0

result: 0200000001faedead95b4dd8d73bc320fcb4f9d43eb4f5e9c2a5d46e577e↙
    ↳ a26448d253d62300000000000ffffffff0000000000
```

Get private key of address (from listunspent for example), then calculate public key with it.

Listing 5.16: Get public key via private key

```
../bitcoin/src/bitcoin-cli -datadir=1 dumpprivkey ↙
    ↳ "miCYAPgb1Z6zsDSFxsKVLYpaQHbBtwxBgi"

use bitaddress at ↙
    ↳ https://www.bitaddress.org/bitaddress.org-v3.3.0-SHA256-dec17c ↙
    ↳ 07685e1870960903d8f58090475b25af946fe95a734f88408cef4aa194↙
    ↳ .html?testnet=true

result compressed: 03↙
    ↳ F726941B696484D4DE0DB72F36FF5AA4E67C8BD7AC129CE78C59BE0DF3BBAC0A
```

Use bitcoin-tx with the cleaned tx and a public key to add a p2pk output to the empty transaction.

Listing 5.17: Remove output 0

```
../bitcoin/src/bitcoin-tx -regtest 0200000001faedead95b4dd8d73bc320f↙
    ↳ cb4f9d43eb4f5e9c2a5d46e577ea26448d253d6230000000000ffffffff0000↙
    ↳ 000000 ↙
    ↳ outpubkey=49.9999:03F726941B696484D4DE0DB72F36FF5AA4E67C8BD7A ↙
    ↳ C129CE78C59BE0DF3BBAC0A

Result: 0200000001faedead95b4dd8d73bc320fcb4f9d43eb4f5e9c2a5d46e577e↙
    ↳ a26448d253d6230000000000ffffffff01f0ca052a01000000232103f726941↙
    ↳ b696484d4de0db72f36ff5aa4e67c8bd7ac129ce78c59be0df3bbac0aac0000↙
    ↳ 0000
```

The raw transaction can be checked by using the human-readable json-output function

Listing 5.18: Check the raw transaction by utilizing the JSON export functionality

```
../bitcoin/src/bitcoin-tx -regtest -json 0200000001faedead95b4dd8d73↙
    ↳ bc320fcb4f9d43eb4f5e9c2a5d46e577ea26448d253d6230000000000ffffff↙
    ↳ ff01f0ca052a01000000232103f726941b696484d4de0db72f36ff5aa4e67c8↙
    ↳ bd7ac129ce78c59be0df3bbac0aac00000000
```

Until now, the transaction will not be accepted by other nodes because it lacks the needed cryptographic signatures. Thus, we create them:

Listing 5.19: Signing the raw transactions

```
../bitcoin/src/bitcoin-cli -datadir=1 signrawtransaction ↙
    ↳ "0200000001faedead95b4dd8d73bc320fcb4f9d43eb4f5e9c2a5d46e577ea ↙
    ↳ 26448d253d6230000000000ffffffff01f0ca052a01000000232103f726941b↙
    ↳ 696484d4de0db72f36ff5aa4e67c8bd7ac129ce78c59be0df3bbac0aac00000↙
    ↳ 000"

result: 0200000001faedead95b4dd8d73bc320fcb4f9d43eb4f5e9c2a5d46e577e↙
    ↳ a26448d253d623000000004948304502210 0a2caf810a461347213b5f47970a↙
    ↳ 0e6b8a20e1bbba925d7bfcf3838d5d4a091aa022075f41466ab318b82228183↙
    ↳ 2764c3b4d44c71e01f218e429af3512e1ee8ddbcdc01ffffffff01f0ca052a0↙
    ↳ 1000000232103f726941b696484d4de0db72f36ff5aa4e67c8bd7ac129ce78c↙
    ↳ 59be0df3bbac0aac00000000
```

Now that the transaction is successfully created, we need the proof that a given public key is not data.

Listing 5.20: Creating the proof

```
../bitcoin/src/bitcoin-cli -datadir=1 createoutputpkproof ↙
    ↳ miCYAPgb1Z6zsDSFxsKVLYpaQHbBtwxBgi

result: {
  "publickey": "en3ALEz6LxNAJPo5wuDKEB1iDRVA1t8cyYyBSZa7y6yg",
  "proofpk": ↙
      ↳ "3sgA3WCrtM9vn6PdtbBer4HPiNU9JYfg9vYP8noeneXYc7UYtb8G3yXdo6C ↙
      ↳ k1myNuZToDUNBXkdqCkPjsV2Ns6pRs"
}
```

The created proof is then imported into both bitcoin core nodes of our regtest-environment.

Listing 5.21: Importing the proof

```
../bitcoin/src/bitcoin-cli -datadir=2 addknownpreimagepk 3↙
    ↳ sgA3WCrtM9vn6PdtbBer4HPiNU9JYfg9vYP8noeneXYc7UYtb8G3yXdo6Ck1my ↙
    ↳ NuZToDUNBXkdqCkPjsV2Ns6pRs en3ALEz6LxNAJPo5wuDKEB1iDRVA1t8cyYy ↙
    ↳ BSZa7y6yg

Result:{
  "Notice": "Successfully added public key proof"
}
```

The proof-consensus-rules added to bitcoin core are not activated by default. The following commands activate them and cause bitcoin core to reject new blocks with missing proofs.

Listing 5.22: Enabling the proof requirement on both Bitcoin nodes

```
../bitcoin/src/bitcoin-cli -datadir=1 enableproofrequired 1
../bitcoin/src/bitcoin-cli -datadir=2 enableproofrequired 1
```

Lastly, we tell our client about the signed transaction. In the process, it is added to the mempool and sent out to other nodes on the network.

Listing 5.23: Publishing the transactions

```
../bitcoin/src/bitcoin-cli -datadir=1 sendrawtransaction 0200000001f↙
    ↳ aedead95b4dd8d73bc320fcb4f9d43eb4f5e9c2a5d46e577ea26448d253d623↙
    ↳ 0000000049483045022100a2caf810a461347213b5f47970a0e6b8a20e1bbba↙
    ↳ 925d7bfcf3838d5d4a091aa022075f41466ab318b822281832764c3b4d44c71↙
    ↳ e01f218e429af3512e1ee8ddbcdc01ffffffff01f0ca052a01000000232103f↙
    ↳ 726941b696484d4de0db72f36ff5aa4e67c8bd7ac129ce78c59be0df3bbac0a↙
    ↳ ac00000000

result: 2e87ea75525f057b424e4e2cd926c70b63f66a5fc0ea5bf9fdadefa60013↙
    ↳ 211a #txid
```

The first node generates a block that includes the published transaction. It automatically sends this block to the second node. The second node successfully accepts the block.

Listing 5.24: Publishing the transactions

```
../bitcoin/src/bitcoin-cli -datadir=1 generate 1
```

5.3 Implementation Details and Source Code Commentary

Creating the proof of concept for the preimage- and public-key-proof solution was a somewhat complicated task. I decided early on to implement the code within the most widely used reference implementation of Bitcoin, Bitcoin Core. It is a project written in C++, and has grown to over 330.000 lines of code within nearly 10 years of being actively developed. Compared to that, our change to Bitcoin Core There were multiple reasons for the choice to implement the solution within Bitcoin Core itself. It allowed me to look at the actual code of the consensus mechanism and other parts of the protocol, without having to trust the validity of a reimplementation. It would also allow me to see potential corner cases presented in the code, something I might not become aware of if I had reimplemented the consensus mechanism according to online documentation sources. The decision to implement the proof of concept also allowed me to be sure that there would be no function interdependencies that I could miss, and that might have made the solution completely unviable. Code is less forgiving than mental models of how a system is supposed to work, and should expose some flaws in reasoning.

That said, only half of the solution could be implemented. Actually implementing the whole solution would require much more code changes, as the transaction layout would require a redesign. Additionally, the network subsystem would have to be extended and made aware of the proofs and new censorship transactions. It would also be more sensible to change a few other parts of Bitcoin before or when implementing this proof of concept for actual usage. For instance, the problem of data-inclusion by causing partial hash collisions in transaction ids might be something that should be solved via the introduction of a different hashing algorithm, as attackers might just opt to abuse this part of the system when it becomes impossible to include arbitrary data inside addresses or public key pushes. As the result for the network is nearly the same, it makes more sense to combine these two big changes, requiring just one bigger change to consensus rules instead of needlessly creating a lot of work for the community to advertise for both changes independently.

When writing the implementation, my decisions on where to add code focused on easy understandability by people unfamiliar with the Bitcoin codebase. For this reason, most actual code resides in two new files that were created to contain the actual proof creation and validation functions. This allows an outsider to read through the changes, and to use code usage searches to find out where these functions come into play in the rest of the code base.

Care was taken to reuse existing functions. This was complicated at first, as documentation of the functions used within Bitcoin Core is often non-existent. Even important functions such as those for signature validation, or those that allow users to access private keys, are generally not commented at all. Implementation details are also not supported by comments, requiring a developer to jump between different functions to figure out when and where certain high-level concepts are actually implemented. Some functions exist in different variants and it can be unclear which one is to be preferred in case of a reimplementation, or how to convert the data to supply it to the function with the proper type. At times, the implementation of the proof of concept caused the software to become unstartable because certain library includes seemed to break the initialization. As my C++ experience is not substantial, this can also be attributed to unfamiliarity with the language, however. This short comment on my experience working on the code base of Bitcoin Core should serve as a heads-up for people who want to attempt a similar project. Ultimately, one has to become familiar with the commonly used functions, and looking at already implemented functions doing a similar task can help to reduce the time needed to implement a wanted function properly.

What follows is a list of the edited/created files, and the purpose of the changes contained within them.

txverify.cpp In txverify, the actual transaction checking is implemented. This means a transactions layout and completeness is checked here. Any check that fails invalidates the whole transaction and allows the client to reject it completely. The proof of concept require a small addition in this file to check for preimage existence in case the feature is activated. If no (previously checked) proof is available, the transaction will be rejected just as if its other cryptographic proofs or layout aren't valid.

blockchain.cpp This file contains the implementation for RPC commands related to the blockchain and to viewing certain transactions or saving certain information to disk. It was chosen as the next best fit for the RPC proof functions that can be called from the command line. RPC commands are added to an array at the end of the file, linking to functions with a specific layout. Requests and replies are encoded in JSON, and there are error handlers in place to catch user errors when calling the functions. Code creating the JSON and error handling are done here, whereas the real code implementing the function that is requested of the user is implemented in preimage.cpp.

preimage.h This is the header file for most of our code. It includes all necessary libraries and declares public and private functions found within preimage.cpp. Among the libraries that our solution needs are the hashing functions to (re)create the addresses, public and private key functions, base58 encoding functions to export data in the commonly used Bitcoin base58 format, and multiple wallet-libraries for the lookup of keys within the users Bitcoin wallet, as those are needed to create the proofs.

preimage.cpp The main file of the proof of concept. Covers the activation and deactivation of the proof requirement, proof creation, proof import, and saving and loading the proofs automatically. The most important functions within here are CreateP2PKProof and CreateP2PKHProof, as they show how the actual creation of a proof is done. The storage functions at the end of the file merely recreate the strings already used for the RPC command output and write them to a file, loading them again when the proof checking is enabled after startup. Other than including some needed libraries, preimage.cpp is implemented without the need for hooks into other portions of the code. Creating the proof does not require anything but the private key of a Bitcoin output, which causes lower coupling and a high cohesion within preimage.cpp. Proof storage and enabling/disabling functions are merely convenience functions for the proof of concept and would probably be removed in a real implementation as disabling the proof requirement would not be necessary anymore.

After this high-level look at these important source files, we will take a closer look at specific functions, explaining how they work.

5.3.1 txverify.cpp

To keep the code properly sorted, the original txverify.cpp was only expanded by the two function calls shown below. They check if the preimage/proof solution is activated, and if it is, they check if all outputs within a transaction have a corresponding preimage or signature.

As an additional side note, the check here should be expanded in a real implementation. A skip for this check should prevent older blocks to be checked for the existence of proofs, so proofs can be forgotten after a while. A good block height to do this would be after

a minimum of 100 blocks, as this is when newly mined coins become spendable. This protects the proof system from a smaller reorg, meaning a chain split that is occurring while the Bitcoin Core software is running and that ends in another blockchain becoming the longest and thus valid one. Short reorgs do not pose a problem as the proofs will not have been deleted yet. On the other hand, reorgs that go back further than 100 blocks cause fundamental problems for Bitcoin, so the theoretical failure to provide evidence beyond 100 blocks would not add much to the more serious problem of the caused financial loss by miners and payment receivers. That said, a larger threshold for saving the proof could also be easily implemented.

Listing 5.25: Publishing the transactions

```
1 bool CheckTransaction(const CTransaction& tx, CValidationState ↙
    ↳ &state, bool fCheckDuplicateInputs)
2 {
3   ... // various checks of transaction validity before this
4     // check output not containing bad data by validating hash ↙
        ↳ preimages/public key!
5     if (PREIMAGE.isEnabled() && !PREIMAGE.HasKnownPreimage(tx)) {
6      return state.DoS(100, false, REJECT_INVALID, "no known address ↙
         ↳ preimage");
7     }
8 }
```

5.3.2 rpc/blockchain.cpp

This file contains the implementation of some RPC commands in regard to the blockchain, the Bitcoin Core mempool, and similar functions. To add commands to the RPC interface of Bitcoin Core, this format has to be followed when adding new commands. It basically matches the entered string to an internal function of the same file, and also names the parameters directly following the function when e.g. entering the command on a command-line-interface.

Listing 5.26: Commands array with newly added proof of concept commands

```
1 static const CRPCCommand commands[] =
2 { //  category              name                        actor ↙
    ↳ (function)         argNames
3
4   // ... other rpc functions concerning blockchain data
5
6  // preimage solution
7  { "blockchain",    "addknownpreimage",    &addknownpreimage,    ↙
     ↳ {"preimage", "address"} },
8  { "blockchain",    "addknownpreimagepk",   &addknownpreimagepk,    ↙
     ↳ {"signature", "publickey"} },
9  { "blockchain",    "createoutputpkhproof",   &createoutputpkhproof, ↙
     ↳   {"txid", "n"} },
10 { "blockchain",    "createoutputpkproof",    &createoutputpkproof, ↙
     ↳   {"address"} },
11 { "blockchain",    "enableproofrequired",    &enableproofrequired, ↙
     ↳    {"enable"} },
12 { "blockchain",    "disableproofrequired",   &disableproofrequired, ↙
     ↳    {"disable"} },
13   ...
14 };
```

It would be bad style to implement the functions in this file directly. For this reason, all the functions that are called in this file only call functions actually implemented within the preimage.cpp source file. Instead, answer formatting and returning error codes are handled here as they are only relevant to the RPC interaction code. As an example, we will take a closer look at the function that creates a proof for an entered public key that we know the secret key to:

Listing 5.27: createoutputpkproof RPC implementation

```
1  UniValue createoutputpkproof(const JSONRPCRequest& request) {
2      if (request.params[0].isNull())
3       throw JSONRPCError(RPC_INVALID_PARAMETER, "Missing address ↙
           ↳ parameter");
4
5      std::string strAddress = request.params[0].get_str();
6      std::vector<unsigned char> proofpk;
7      std::vector<unsigned char> publickey;
8
9    PREIMAGE.CreateP2PKProof(strAddress, proofpk, publickey);
10
11     UniValue ret(UniValue::VOBJ);
12     ret.push_back(Pair("publickey", EncodeBase58(publickey)));
13     ret.push_back(Pair("proofpk", EncodeBase58(proofpk)));
14
15     return ret;
16 }
```

After checking that an address is supplied, we call the function within the preimage module that creates a proof for it. It is then formatted to match the return format for the RPC subsystem, reincluding the supplied public key for a better user interaction when having called the function.

As already mentioned, all actual work is done within preimage.cpp. Only parameter extraction and data conversion have to be handled by blockchain.cpp. The next section thus covers the actual implementation of the important functions.

5.3.3 preimage.cpp

The file preimage.cpp was created for this thesis and only includes code written for the proof system. Explaining every function in detail would extend the scope of this master thesis. However, exposure to some details of actual code implementations should be benefitial in case the proof of concept is studied in more detail. A good function to see how proofs are implemented is the public key proof creation function.

The comments in the file should give an overview of what every part is meant to do. Some functions will be discussed in more detail after the full code listing.

Listing 5.28: createoutputpkproof RPC implementation

```
1
2  bool preimage::CreateP2PKProof(std::string strAddress, valtype& ↙
       ↳ proof, valtype& publickey) {
3  // 1. get hash
4  CWallet * const pwallet = vpwallets[0];
5  uint160 hash;
6  // 2. find the private key to the public key
7  CTxDestination dest = DecodeDestination(strAddress);
8  const CKeyID *keyID2 = boost::get<CKeyID>(&dest);
9  if (!keyID2) {
```

```
10  return false;
11 }
12 CKeyID keyID = *keyID2;
13
14 CKey vchSecret;
15
16 pwallet->GetKey(keyID, vchSecret);
17 CKey key = CBitcoinSecret(vchSecret).GetKey();
18
19 // 3. extract the public key again to have it in the right format f↙
       ↪ or signature testing
20 CPubKey pubkey = key.GetPubKey();
21
22 // 4. create the hash of the public key. This is the text that will ↙
       ↪ be signed with the private key
23 unsigned char* csha256new = new unsigned char[32]();
24 CSHA256().Write(pubkey.begin(), pubkey.size()).Finalize(csha256new);
25 std::vector<unsigned char> vec(csha256new, csha256new+32);
26 uint256 pubkey_hash = uint256(vec);
27
28 // 5. create the signature and store it in vch_signature. Abort if a↙
       ↪ ny problems happen
29 valtype vch_signature;
30 if (!key.SignCompact(pubkey_hash, vch_signature)) {
31  return false;
32 }
33
34 // 6. RecoverCompact is the signature verification function for ↙
       ↪ text signed with a private key. We test our newly created ↙
       ↪ signature here to be sure it was created successfully
35 if (!pubkey.RecoverCompact(pubkey_hash, vch_signature)) {
36  return false;
37 }
38
39 // 7. at this point, the signature is created and verified. all ↙
       ↪ that is left is to give it back to the caller
40 proof = vch_signature;
41 valtype newpub(pubkey.begin(), pubkey.end());
42 publickey = newpub;
43
44 // 8. if nothing has returned an error until now, the function ↙
       ↪ returns positively
45 return true;
46 }
```

Function parameters CreateP2PKProof needs to know what public key it should create a proof for. There are multiple ways to design a function that makes this easy for the end-user. In this case, the address string of a public key was used, because end-users are generally exposed to them already.

Another way this could have been implemented is by referring to an already created transaction and creating all necessary proofs for this transaction only. This has also been done, but for the other type of proof. The function parameter in this case is just the string the end-user supplied to the RPC command. Additionally, two pointers are supplied to the function: One to return the public key to the caller, and another one for the proof, e.g. the created signature.

Line 4 The wallet containing the private keys of a node is prepared.

Line 7-17 Using the address of the public key, the correct keyId is deducted. This keyId is then used in line 17 to get the actual secret key by querying the CWallet that was created previously.

Line 23-26 This part of the code highlights the common usage of the hashing function for SHA256 within Bitcoin. Many functions encapsulate this functionality, e.g. when creating IDs for transactions, but we need to access it directly in some instances to access the preimage, for example.

CreateP2PKProof could be implemented differently and sign a fixed text instead of the hash of the public key. The hash was chosen because it can be easily recreated on other nodes and because hashes are often used as targets for signature algorithms, making implementation issues more unlikely.

Line 30 The actual signature creation happens here. The function "SignCompact" is supplied with the message (the hash of the public key) and a data structure than can store the signature. If an error is encountered, the function returns false, aborting the CreateP2PKProof function by having it return false, too.

Line 35 Finding the function "RecoverCompact" that does signature verification was not easy. The Bitcoin Core source code is often lacking in comments, so only careful reading of the code allowed the author to find this function. We call this function here to check that the signature that was just created is actually valid. This should always be the case, but considering that the function here is only meant to be called when a user interacts with the Bitcoin Core code, the additional resource consumption is absolutely negligible.

Line 40-45 The proof and pubkey are written to the supplied locations and the function returns positively.

A similar function is CreateP2PKHProof. This function does not create a public key proof, but a proof for a public key hash. By calculating the first hash of the public key (line 20-23), the preimage for the address can be determined, as the process of Bitcoin address creation involves two hash iterations of the public key. Hashing once does thus provide the preimage for the address that is later included in the Bitcoin output.

Listing 5.29: CreateP2PKProof implementation

```
1  // Creates proof for a public key hash as used in a P2PKH transaction
2  bool preimage::CreateP2PKHProof(valtype vch_pkh, uint256& ↙
       ↳ preimageproof) {
3   // 1. get hash
4      uint160 hash = uint160(vch_pkh);
5
6   // 2. find corresponding key in wallet
7      CWallet * const pwallet = vpwallets[0];
8
9      CKeyID keyID = CKeyID(hash);
10
11     CKey vchSecret;
12
13     if (!pwallet->GetKey(keyID, vchSecret)) {
14      return false; //Private key for address strAddress is not known
15     }
16     CPubKey pubkey = CBitcoinSecret(vchSecret).GetKey().GetPubKey();
17
18
```

```
19    // 3. hash public key once with SHA256(!) to get proof
20      unsigned char* csha256new = new unsigned char[32]();
21      CSHA256().Write(pubkey.begin(), ↙
            ↳ pubkey.size()).Finalize(csha256new); // finally contains ↙
            ↳ the correct hash \o/
22      std::vector<unsigned char> vec(csha256new, csha256new+32);
23      uint256 pubkey_hash = uint256(vec);
24
25      preimageproof = pubkey_hash;
26
27      return true;
28  }
```

The last major code example shows how we detect P2PK and P2PKH outputs and check for the existence of a proper proof. The caller supplies an output within a transaction, and we iterate through all operations of the script of this output. When the code encounters a push command for 20 (= hash/address, line 19) or 33 bytes (= public key, line 25), the corresponding check is called to see if a proof is available.

Listing 5.30: Checking outputs for preimages/proofs

```
1  // Checks if a transaction output's proofs are available. This f↙
       ↳ unction should be called for standard transactions only. This f↙
       ↳ unction mimics the code of EvalScript in interpreter.cpp.
2  bool preimage::HasKnownPreimage(const CTxOut& output) {
3
4      CScript script = output.scriptPubKey;
5      CScript::const_iterator pc = script.begin();
6      CScript::const_iterator pend = script.end();
7      CScript::const_iterator pbegincodehash = script.begin();
8      opcodetype opcode;
9      valtype vchPushValue;
10     std::vector<valtype> altstack;
11
12     try
13     {
14         while (pc < pend)
15         {
16          // filling opcode
17          script.GetOp(pc, opcode, vchPushValue);
18          // opcode 20 => push of the bytes of a ripemd160-hash, the ↙
                ↳ hash type used in p2pkh
19          if (opcode == 20) {
20            if (IsHashWithKnownPreimage(vchPushValue) == false) {
21              return false;
22            }
23          }
24          // opcode 33 => push of a (compressed) public key
25          else if (opcode == 33) {
26            if (IsPublicKeyWithProof(vchPushValue) == false) {
27              return false;
28            }
29          }
30         }
31     }
32     catch (...)
33        { // error -> return false to be sure;
34       return false;
35        }
```

```
36     return true; // no error encountered. Checks were passed, ↙
           ↳ preimages/proofs are available for the output.
37 }
```

Other functions behave similarly. Functions within Bitcoin Core seem to require frequent conversion of data when new unexpected functions are to be implemented. This lack of easily usable conversion functions is not surprising considering that most of Bitcoin Core is security critical code where faulty changes could have catastrophic results. In case this proof of concept was reimplemented, it might be reasonable to expand commonly used objects with such conversion functions.

5.3.4 preimage.h

As is typical for header files, function definitions and includes can be found here. One important section is right at the beginning of the file, however:

Listing 5.31: createoutputpkproof RPC implementation

```
1 #ifndef BITCOIN_PREIMAGE_H
2 #define BITCOIN_PREIMAGE_H
```

Without this declaration, the preimage code would be compiled multiple times by the Bitcoin code base, breaking the compilation run in the process. If future implementations of the preimage functionality were to split into more than one new code file, this should be taken into account, as tracking down the error that is created by missing these two lines can be exhausting.

This concludes the implementation details of the Bitcoin Core proof of concept code. Besides this explanation of code details, the comments that were added to the code should serve as a guide when trying to understand the exact implementation decisions whe writing this modification.

6. Conclusion

This chapter summarizes the thesis. It recaps the approaches taken to further protect Bitcoin from off-topic data inclusion, and describes future work that will be relevant.

6.1 Summary

The thesis covers a specific problem within the sphere of cryptocurrencies, and especially Bitcoin. Designed as a barrier less network for online payments, without the need for any particular rule, abuse and misuse potential is potentially higher than for centralized payment systems. It is further designed with low barriers for entry, allowing everybody to participate without a need to ask for permission from anyone in particular. However, most of its rules only cover parts essential for the technical and secure functioning of the system, and the project's aim to be censorship resistant allows for some attack vectors that are difficult to protect against, as they do not violate any of the current rules that are enforced by the software itself.

The problem that was presented is the possibility to include arbitrary and thus potentially illegal or immoral data by hiding it within monetary transactions. The problem is amplificated by the open design of Bitcoin which allows users to essentially make up their own rules for how the recipient has to prove ownership of his newly received bitcoins. Even standard transactions pose a risk of being usable for covert data inclusion, however. As these techniques have been used on the main Bitcoin network for benign files, it is apparent that the risk for potential abuse is not just theoretical. For this reason, this thesis looked into prior work on how to include hard-to-remove data within permission less computer networks, and potential ways to make them infeasible.

It also takes a close look at the concrete implementation of Bitcoin in an attempt to validate the usefulness of the approaches that are meant to harden the network. This is done as some more straight-forward attempts also discussed in this thesis seem to only focus on solving the problem in a more general case that does not focus on the unique propertiy of permissionlessness of blockchains like Bitcoin. While it can certainly be challenging to implement deletion capabilities in more convoluted data structures, the problem is generally exaggerated by not having a benevolent single control instance such as a company. Introducing such a control instance to solve the data inclusion problem seems antithetic to the foundations of Bitcoin, which has been devised and improved with the goal of further decentralization and resilience against being controllable by one or few actors.

For this reason, this thesis tried to present approaches that keep the old system as intact as possible, and essentially allows every participant to check for themselves if the new rules that would be imposed upon Bitcoin have been carried out successfully. The first solution covers the inclusion of data within commonly used transaction types, making it harder to hide data within the receiving-address-portion of a transaction. The second solution then covers Bitcoins more advanced feature-set made possible via Bitcoin Script. By making a small part of non-standard transactions censorable if they do not provide non-data-inclusion proof for every data push, a compromise between eternal data storage and the integrity of the whole system is achieved. This compromise keeps Bitcoin's most important characteristics such as inflation resistance intact, and abuse of the system does cause no monetary harm to the person whose transaction has been partly censored.

The first solution has been coded as an addon to the most popular Bitcoin node software, bitcoin core, and Chapter describes how the proof of concept implementation can be used to test the concept that prevents data inclusions into addresses or the public key portion of a transaction. This was done to show that parts of the complete solution can be implemented without a complete reorganization of the project, and that the approach is not hindered by implementation details of Bitcoin that might escape a cursory glance on how the protocols involved operate. It might also serve as a guidance for proper implementation on the main Bitcoin project, although it is more likely that it would be immediately added to the responsible code parts, instead of being bundled within an extra file to make it easier to understand for the purpose of understanding this thesis. In any case, as Bitcoin lacks a central ruler, an agreed-upon activation mechanism and worldwide consensus on implementing the change would be required. It is outside the scope of this thesis on how this would be achieved, but more recent developments show how it could be done from a technical perspective. It could even be phased in like other updates, allowing Bitcoin miners to decide for themselves if they want to risk creating a block containing illegal data.

The thesis concludes with some details surrounding the code that was written to create the solution.

6.2 Future Work

Bitcoin and other cryptocurrencies are an exciting new field of study. While writing this thesis, multiple new developments have come to the attention of the technical community surrounding Bitcoin. While details are often scarce and proposed solutions lack an implementation example at first, a general trend towards even more feature rich cryptographic functions could be observed.

Improvements such as Schnorr functions, which would serve as a replacement for the currently used ECDSA cryptography routines, have attributes that might make them extremely useful for further reduction of data inclusion vulnerabilities. As an example, the aggregation of signatures that is possible with Schnorr reduces the size of transactions, while retaining the non-interactivity of the process of signing transactions. While information is hard to come by, this could allow for a scheme where well-known entities provide a service that adds their signature to any Bitcoin transaction that wishes it, essentially making the meaningful portion of a transaction non-attacker controlled any more.

Similarly, a technique known as Merklized Abstract Syntax Trees (MAST) would allow for the hiding of payment requirements (i.e. smart contract data) until it is acted upon, which would make it impossible to include bad data within conditional branches of smart contracts that are never used, as those would not be published to the blockchain in any case. As a further restriction it might be possible to remove certain opcodes from Bitcoin

Script that allow script portions to be disregarded, especially since there might be a way to reimplement those with MAST-enabled conditional statements. This would not reduce the power of Bitcoin Script, while further hardening Bitcoin against data inclusion.

This thesis also mentioned some less effective schemes for data inclusion that might become used if all remaining effective options for data inclusion are removed. Protecting against those seems to require even further improvements to the basics of Bitcoin, such as a replacement for the utilized hashing functions, or a complete overhaul of how transactions work. One such proposal for an overhaul has been released anonymously under the name of MimbleWimble. It uses zero knowledge proofs to hide transaction amounts from people not involved with a transaction, which stops attackers from including data by encoding it into the least significant bits of the amount field of Bitcoin transaction outputs. Attackers now are very unlikely to utilize it as other schemes offer significantly more space for a vastly reduced transaction fee, but more research will be needed to fix this issue once all other possibilities have been eradicated, as sending an amount of a monetary transaction unit surely can't be removed from Bitcoin without destroying its primary main feature.

When defining the problem space of this thesis, some methods of data inclusion were deemed outside of the scope of this work. To stop them from working, a scheme with a hash function that makes partial hash collissions even more computationally intensive might prove useful. It is unclear, however, how this might affect block validation, and it is further unclear how this might affect block propagation and other metrics on the Bitcoin peer to peer network. For this reason, further research and benchmarking of such changes to a test network might prove very useful.

Looking at the broader picture, the topic of data insertion into uncensorable data structures does also present novel problems for the legal system, and it remains to be seen if a compromise can be found that might even allow illegal information to exist within such a system, as long as it is only used for the intended, legal use of the technology. Similarly to how internet providers cannot be held accountable for crimes committed through their internet access, running a node and not accessing certain portions of a blockchain but to confirm their validity could be seen as legal, as it lacks the intent of committing a crime in most cases. Still, it is not impossible that some part of the solution will involve techniques that e.g. make it easier to prove that there was no attempt at obtaining or spreading illegal data. The cryptocurrency Ethereum, for example, is working on a scheme that is called sharding. With sharding, block verification is spread between multiple parties. One could imagine implementing a sharding algorithm so that a single node operator only ever obtains and shares a small part of an illegal data file, making them not guilty of possession or spreading of the data because the part is deemed too insignificant to be of concern. It would also enable law enforcement to differentiate between honest miners, and people using the blockchain to aquire the illegal data. Whether such a compromise is good enough, and if sharding is an option for Bitcoin or destroys one of its important properties, remains to be seen, however.

Bibliography

[AMVA17] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on.* IEEE, 2017, pp. 111–126.

[And12] G. Andresen, "Bip 0016: Pay to script hash," 2012.

[BBD+11] J. Backes, M. Backes, M. Dürmuth, S. Gerling, and S. Lorenz, "X-pire!-a digital expiration date for images in social networks," *arXiv preprint arXiv:1112.2649*, 2011.

[bit18a] bitcoin.it, "Eligius," 2018, [accessed August 17, 2018]. [Online]. Available: https://en.bitcoin.it/wiki/Eligius

[bit18b] bitcoin.org, "en-soft-fork.svg," 2018, [accessed August 17, 2018]. [Online]. Available: https://bitcoin.org/en/developer-guide#consensus-rule-changes

[bit18c] ——, "en-unlocking-p2sh-output.svg," 2018, [accessed August 17, 2018]. [Online]. Available: https://bitcoin.org/en/developer-guide#p2sh-scripts

[bit18d] ——, "Standard p2pkh pubkey script," 2018, [accessed August 17, 2018]. [Online]. Available: https://bitcoin.org/en/developer-guide#p2pkh-script-validation

[Eno11] Enochian, "Re: New attack vector," 2011, [accessed August 19, 2018]. [Online]. Available: https://bitcointalk.org/index.php?topic=8392.msg123728#msg123728

[ES14] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International conference on financial cryptography and data security.* Springer, 2014, pp. 436–454.

[Fel08] W. Feller, *An introduction to probability theory and its applications.* John Wiley & Sons, 2008, vol. 2.

[GG11] P. Güring and I. Grigg, "Bitcoin & gresham's law-the economic inevitability of collapse," *October–December. http://iang.org/papers/BitcoinBreachesGreshamsLaw.pdf*, 2011.

[GKL15] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications." in *EUROCRYPT (2)*, 2015, pp. 281–310.

[GKLL09] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data." in *USENIX Security Symposium*, vol. 9, 2009.

[GS15] G. Gutoski and D. Stebila, "Hierarchical deterministic bitcoin wallets that tolerate key leakage," in *International Conference on Financial Cryptography and Data Security.* Springer, 2015, pp. 497–504.

[Lav18] S. Lavine, "Bitcoin testnet box," 2018, [accessed August 16, 2018]. [Online]. Available: https://github.com/freewil/bitcoin-testnet-box/

[Max18a] G. Maxwell, "Confidential values," 2018, [accessed August 17, 2018]. [Online]. Available: https://people.xiph.org/~greg/confidential_values.txt

[Max18b] ——, "P2sh^2," 2018, [accessed August 19, 2018]. [Online]. Available: https://web.archive.org/web/20150701093818/http://sourceforge.net:80/p/bitcoin/mailman/message/30705609/

[Mer80] R. Merkle, "Protocols for public key cryptosystems„" in *1980 Symposium on Security and Privacy*. IEEE Computer Society, 1980, pp. 122–133.

[MHH+18] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, D. Müllmann, O. Hohlfeld, and K. 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). Springer*, 2018.

[MHZ+18] R. Matzutt, M. Henze, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Thwarting unwanted blockchain content insertion," in *Cloud Engineering (IC2E), 2018 IEEE International Conference on*. IEEE, 2018, pp. 364–370.

[MLS+15] E. McReynolds, A. Lerner, W. Scott, F. Roesner, and T. Kohno, "Cryptographic currencies from a tech-policy perspective: Policy issues and technical directions," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 94–111.

[Nak08] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[Nar15] A. Narayanan, "Analyzing the 2013 bitcoin fork," 2015, [accessed August 11, 2018]. [Online]. Available: https://freedom-to-tinker.com/2015/07/28/analyzing-the-2013-bitcoin-fork-centralized-decision-making-saved-the-day/

[PDC17] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to forget without hard forks." *IACR Cryptology ePrint Archive*, vol. 2017, p. 106, 2017.

[RH11] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 1318–1326.

[Ros11] M. Rosenfeld, "Analysis of bitcoin pooled mining reward systems," *arXiv preprint arXiv:1112.4980*, 2011.

[Tow16] A. Towns, "Segregated witness benefits," 2016, [accessed July 22, 2018]. [Online]. Available: https://bitcoincore.org/en/2016/01/26/segwit-benefits/

[tra15] tradeblock.com, "Analysis of bitcoin transaction size trends," 2015, [Online; accessed December 10, 2017]. [Online]. Available: https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends

[Var16] b. Various, "Prohibited changes," 2016, [Online, accessed December 15, 2017]. [Online]. Available: https://en.bitcoin.it/wiki/Prohibited_changes

[WHH+10] S. Wolchok, O. S. Hofmann, N. Heninger, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel, "Defeating vanish with low-cost sybil attacks against large dhts." in *NDSS*, 2010.