Redactable Blockchain in the Permissionless Setting

Dominic Deuber*, Bernardo Magri† and Sri Aravinda Krishnan Thyagarajan*
*Friedrich-Alexander University Erlangen-Nürnberg, Germany
E-mail: {deuber, thyagarajan}@cs.fau.de
† Aarhus University, Denmark
E-mail: magri@cs.au.dk

***Abstract*—Bitcoin is an immutable permissionless blockchain system that has been extensively used as a public bulletin board by many different applications that heavily relies on its immutability. However, Bitcoin's immutability is not without its fair share of demerits. Interpol exposed the existence of harmful and potentially illegal documents, images and links in the Bitcoin blockchain, and since then there have been several qualitative and quantitative analysis on the types of data currently residing in the Bitcoin blockchain. Although there is a lot of attention on blockchains, surprisingly the previous solutions proposed for data redaction in the permissionless setting are far from feasible, and require additional trust assumptions. Hence, the problem of harmful data still poses a huge challenge for law enforcement agencies like Interpol (Tziakouris, IEEE S&P'18).**

We propose the first efficient redactable blockchain for the permissionless setting that is easily integrable into Bitcoin, and that does not rely on heavy cryptographic tools or trust assumptions. Our protocol uses a consensus-based voting and is parameterised by a policy that dictates the requirements and constraints for the redactions; if a redaction gathers enough votes the operation is performed on the chain. As an extra feature, our protocol offers public verifiability and accountability for the redacted chain. Moreover, we provide formal security definitions and proofs showing that our protocol is secure against redactions that were not agreed by consensus. Additionally, we show the viability of our approach with a proof-of-concept implementation that shows only a tiny overhead in the chain validation of our protocol when compared to an immutable one.

***Index Terms*—Blockchain, Bitcoin, Redactable Blockchain, GDPR**

I. Introduction

Satoshi Nakamoto's 2008 proposal of Bitcoin [38] has revolutionised the financial sector. It helped realise a monetary system without relying on a central trusted authority, which has since then given rise to hundreds of new systems known as cryptocurrencies. Interestingly however, a closer look into the basics of Bitcoin sheds light on a new technology, blockchains. Ever since, there has been a lot of ongoing academic research [21], [28], [14], [16] on the security and applications of blockchains as a primitive. A blockchain in its most primitive form is a decentralised chain of agreed upon blocks containing timestamped data.

A consensus mechanism supports the decentralised nature of blockchains. There are different types of consensus mechanisms that are based on different resources, such as Proof of Work (PoW) based on computational power, Proof of Stake (PoS) based on the stake in the system, Proof of Space based on storage capacity, among many others. Typically, users in the system store a local copy of the blockchain and run the consensus mechanism to agree on a unified view of the blockchain. These mechanisms must rely on non-replicability of resources to be resilient against simple sybil attacks where the adversary spawn multiple nodes under his control.

Apart from its fundamental purpose of being a digital currency, Bitcoin exploits the properties of its blockchain, as in being used as a tool for many different applications, such as timestamp service [23], [22], to achieve fairness and correctness in secure multi-party computation [9], [7], [15], [31], and to build smart contracts [30]. It acts as an immutable public bulletin board, supporting the storage of arbitrary data through special operations. For instance, the `OP_RETURN` code, can take up to 80 bytes of arbitrary data that gets stored in the blockchain. With no requirement for centralised trust and its capability of supporting complex smart contracts, communication through the blockchain has become practical, reasonably inexpensive and very attractive for applications.

Blockchain and Immutability. The debate about the immutability of blockchain protocols has gained worldwide attention lately due to the adoption of the new General Data Protection Regulation (GDPR) by European states. Several provisions of the GDPR regulation are inherently incompatible with current permissionless immutable blockchain proposals (e.g., Bitcoin and Ethereum) [26] as it is not possible to remove any data (addresses, transaction values, timestamp information) that has stabilised[1] in the chain in such protocols. Since permissionless blockchains are completely decentralised and allow for any user to post transactions to the chain for a small fee, malicious users can post transactions to the system containing illegal and/or harmful data, such as (child) pornography, private information or stolen private keys, etc. The existence of such illicit content was first reported in [2] and has remained a challenge for law enforcement agencies like Interpol [47]. Moreover, quantitative analysis in the recent work of Matzutt et al. [34] shows that it is not feasible to "filter" all data from incoming transactions to check for malicious contents before the transaction is inserted into the chain. Therefore, once it becomes public knowledge that malicious data was inserted (and has stabilised) into the chain,

†Work done while the author was affiliated with Friedrich-Alexander University Erlangen-Nürnberg.

[1]A transaction (or data) is considered stable in the blockchain when it is "deep" enough into the chain. We formally define this property in Section II-B.

the honest users are faced with the choice of either, willingly broadcast illicit (and possibly illegal [34], [5]) data to other users, or to stop using the system altogether.

This effect greatly hinders the adoption of permissionless blockchain systems, as honest users that are required to comply with regulations, such as GDPR, are forced to withdraw themselves from the system if there is no recourse in place to deal with illicit data inserted into the chain.

A. State of the Art

Specifically to tackle the problem of arbitrary harmful data insertions in the blockchain, the notion of redacting the contents of a blockchain was first proposed by Ateniese et al. [13]. The authors propose a solution more focused on the permissioned blockchain setting[2] based on chameleon hashes [18]. In their protocol, a chameleon hash function replaces the regular SHA256 hash function when linking consecutive blocks in the chain. When a block is modified, a collision for the chameleon hash function can be efficiently computed (with the knowledge of the chameleon trapdoor key) for that block, keeping the state of the chain consistent after arbitrary modifications.

In a permissioned setting where the control of the chain is shared among a few semi-trusted parties, the solution from [13] is elegant and works nicely, being even commercially adopted by a large consultancy company [4], [3], [11]. However, in permissionless blockchains such as Bitcoin, where the influx of users joining and leaving the system is ever changing and without any regulation, their protocol clearly falls short in this scenario, as their techniques of secret sharing the chameleon trapdoor key and running a MPC protocol to compute a collision for the chameleon hash function do not scale to the thousands of users in the Bitcoin network. Moreover, when a block is removed in their protocol it is completely unnoticeable to the users, leaving no trace of the old state. Although this could make sense in a permissioned setting, in a permissionless setting one would like to have some public accountability as to when and where a redaction has occurred.

Later, Puddu et al. [42] proposed a blockchain protocol where the sender of a transaction can encrypt alternate versions of the transaction data, known as "mutations"; the only unencrypted version of the transaction is considered to be the active transaction. The decryption keys are secret shared among the miners, and the sender of a transaction establishes a mutation policy for his transaction, that details how (and by whom) his transaction is allowed to be mutated. On receiving a mutate request, the miners run a MPC protocol to reconstruct the decryption key and decrypt the appropriate version of the transaction. The miners then publish this new version as the active transaction. In case of permissionless blockchains, they propose the usage of voting for gauging approval based on computational power. However, in a permissionless setting a malicious user can simply not include a mutation for his transaction, or even set a mutation policy where only he himself is able to mutate the transaction. Moreover, to tackle transaction consistency, where a mutated transaction affects other transactions in the chain, they propose to mutate all affected transactions through a cascading effect. This however, completely breaks the notion of transaction stability, e.g., a payment made in the past to a user could be altered as a result of this cascading mutation. The proposal of [42] also suffers from scalability issues due to the MPC protocol used for reconstructing decryption keys across different users.

It is clear that for a permissionless blockchain without centralised trust assumptions, a practical solution for redacting harmful content must refrain from employing large-scale MPC protocols that hinders the performance of the blockchain. It also must accommodate public verifiability and accountability such that rational miners are incentivised to follow the protocol.

B. Our Contributions

Editable Blockchain Protocol. We propose the first editable blockchain protocol for permissionless systems in Section III, which is completely decentralised and does not rely on heavy cryptographic primitives or additional trust assumptions. This makes our protocol easily integrable in systems like Bitcoin (as described in Section V). The edit operations can be proposed by any user and they are voted in the blockchain through consensus; the edits are only performed if approved by the blockchain policy (e.g., voted by the majority). The protocol is based on a PoW consensus, however, it can be easily adapted to any consensus mechanism, since the core ideas are inherently independent of the type of consensus used. Our protocol also offers accountability for edit operations, where any edit in the chain can be publicly verified.

Formal Analysis. We build our protocol on firm theoretical grounds, as we formalise all the necessary properties of an editable blockchain in Section IV, and later show that our generic protocol of Section III-C satisfies these properties. We borrow the fundamental properties of a secure blockchain protocol from [21] and adapt them to our setting.

Implementation. We demonstrate the practicality of our protocol with a proof-of-concept implementation in Python. We first show in Section VI that adding our redaction mechanism incurs in just a small overhead for chain validation time compared to that of the immutable protocol. Then, we show that for our protocol the overhead incurred for different numbers of redactions in the chain against a redactable chain with no redactions is minimal (less than 3% for $5,000$ redactions on a $50,000$ blocks chain). Finally, we analyse the effect of the parameters in our protocol by measuring the overhead introduced by different choices of the system parameters when validating chains with redactions.

C. Our Protocol

Our protocol extends the immutable blockchain of Garay et al. [21] to accommodate for edit operations in the following

[2]The permissioned blockchain setting is when there is a trusted third party (TTP) that deliberates on the users' entry into the system.

way: We extend the block structure to accommodate another copy of the transaction's Merkle root, that we denote by *old state*. We also consider an editing policy for the chain, that determines the constraints and requirements for approving edit operations. To edit a block in the chain, our protocol (Fig. 1) executes the following steps:

a) A user first proposes an edit request to the system. The request consists of the index of the block he wants to edit, and a candidate block to replace it.
b) When miners in the network receives an edit request, they first validate the candidate block using its *old state* information and verifying the following conditions: (1) it contains the correct information about the previous block, (2) it has solved the proof of work and (3) it does not invalidate the next block in the chain. If the candidate block is valid, miners can vote for it during the request's voting period by simply including the hash of the request in the next block they mine. The collision resistance property of the hash function ensures that a vote for an edit request cannot be considered as a vote for any other edit request.
c) After the voting period for a request is over, everyone in the network can verify if the edit request was approved in accordance to the policy (e.g., by checking the number of votes it received). If the request was approved, then the edit operation is performed by replacing the original block with the candidate block.

To validate an edited chain, the miners validate each block exactly like in the immutable protocol; if a "broken" link is found between blocks, the miner checks if the link still holds for the *old state* information[3]. In the affirmative case, the miner ensures that the edited block has gathered enough votes and is approved, according to the policy of the chain.

The process of a redaction in our generic protocol as described in Fig. 2 is pictorially presented in Fig. 1.

II. Preliminaries

Throughout this work we denote by $\kappa \in \mathbb{N}$ the security parameter and by $a \leftarrow \mathcal{A}(\mathsf{in})$ the output of an algorithm $\mathcal{A}$ on input in. We also use the terms "redact" and "edit" interchangeably in this paper.

A. Blockchain Basics

We make use of the notation of [21] to describe a blockchain. A block is a triple of the form $B := \langle s, x, ctr \rangle$, where $s \in \{0,1\}^\kappa$, $x \in \{0,1\}^*$ and $ctr \in \mathbb{N}$. Here s is the state of the previous block, x is the data and ctr is the proof of work of the block. A block B is *valid* iff

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

Here, $H : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ and $G : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ are cryptographic hash functions, and the parameter $D \in \mathbb{N}$ is the block's difficulty level.

[3]A similar technique is used in [10] to "scar" a block that was previously redacted.

The blockchain is simply a chain (or sequence) of blocks, that we call $\mathcal{C}$. The rightmost block is called the head of the chain, denoted by $\mathsf{Head}(\mathcal{C})$. Any chain $\mathcal{C}$ with a head $\mathsf{Head}(\mathcal{C}) := \langle s, x, ctr \rangle$ can be extended to a new longer chain $\mathcal{C}' := \mathcal{C}||B'$ by attaching a (valid) block $B' := \langle s', x', ctr' \rangle$ such that $s' = H(ctr, G(s,x))$; the head of the new chain $\mathcal{C}'$ is $\mathsf{Head}(\mathcal{C}') := B'$. A chain $\mathcal{C}$ can also be empty, and in such a case we let $\mathcal{C} := \varepsilon$. The function $\mathsf{len}(\mathcal{C})$ denotes the length of a chain $\mathcal{C}$ (i.e., its number of blocks). For a chain $\mathcal{C}$ of length n and any $q \geq 0$, we denote by $\mathcal{C}^{\lceil q}$ the chain resulting from removing the q rightmost blocks of $\mathcal{C}$, and analogously we denote by ${}^{q\rceil}\mathcal{C}$ the chain resulting in removing the q *leftmost* blocks of $\mathcal{C}$; note that if $q \geq n$ (where $\mathsf{len}(\mathcal{C}) = n$) then $\mathcal{C}^{\lceil q} := \varepsilon$ and ${}^{q\rceil}\mathcal{C} := \varepsilon$. If $\mathcal{C}$ is a prefix of $\mathcal{C}'$ we write $\mathcal{C} \prec \mathcal{C}'$. We also note that the difficulty level D can be different among blocks in a chain.

B. Properties of a Secure Blockchain

In this section we detail the relevant aspects of the underlying blockchain system that is required for our protocol.

We consider time to be divided into standard discrete units, such as minutes. A well defined continuous amount of these units is called a *slot*. Each slot sl_l is indexed for $l \in \{1, 2, 3, \ldots\}$. We assume that users have a synchronised clock that indicates the current time down to the smallest discrete unit. The users execute a distributed protocol to generate a new block in each slot, where a block contains some data. We assume the slots' real time window properties as in [28]. In [21], [39], [28] it is shown that a "healthy" blockchain must satisfy the properties of *persistence* and *liveness*, which intuitively guarantee that after some time period, all honest users of the system will have a consistent view of the chain, and transactions posted by honest users will eventually be included. We informally discuss the two properties next.

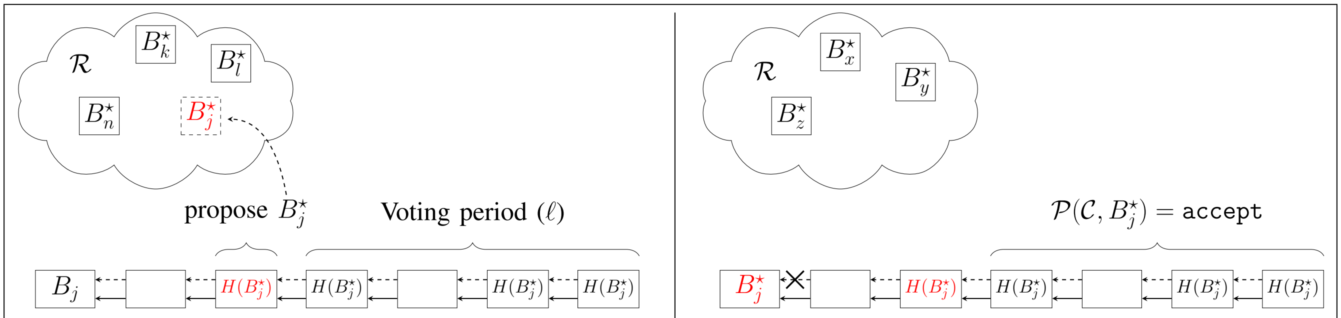
Persistence: Once a user in the system announces a particular transaction as *stable*, all of the remaining users when queried will either report the transaction in the same position in the ledger or will not report any other conflicting transaction as stable. A system parameter k determines the number of blocks that stabilise a transaction. That is, a transaction is stable if the block containing it has at least k blocks following it in the blockchain. We only consider a transaction to be in the chain after it becomes stable.

Liveness: If all the honest users in the system attempt to include a certain transaction into their ledger, then after the passing of time corresponding to u slots which represents the *transaction confirmation time*, all users, when queried and responding honestly, will report the transaction as being stable.

Throughout the paper we refer to the user as both a user and a miner interchangeably.

C. Execution Model.

In the following we define the notation for our protocol executions. Our definitions follow along the same lines of [41].



(a) Proposing a redaction $B_j^\star$ for the block B_j

(b) After a successful voting phase, $B_j^\star$ replaces B_j in the chain

Figure 1: The candidate block pool $\mathcal{R}$ stores the candidate blocks that are proposed and that can be endorsed in the voting phase. A block is linked to its predecessor by two links, the old link (solid arrow) and the new link (dashed arrow). In (a), a redact request $B_j^\star$ is proposed as a redaction for B_j and added to $\mathcal{R}$, then the hash of $B_j^\star$ is included in the chain to denote a new candidate redaction; its voting phase starts just after its proposal. In (b), the candidate block $B_j^\star$ has gathered enough votes and was approved by the redaction policy $\mathcal{P}$ of the chain; $B_j^\star$ replaces B_j and the redacted chain is propagated. Note that new link from the block to the right of $B_j^\star$ is broken (marked by a cross), however the old link to $B_j^\star$ still holds. For simplicity, we consider the parameters $k = 0$ (persistence), $\ell = 4$ (voting period) and $\rho \geq 3/4$ (threshold for policy approval).

A protocol refers to an algorithm for a set of interactive Turing Machines (also called nodes) to interact with each other. The execution of a protocol Π that is directed by an environment/outer game $\mathcal{Z}(1^\kappa)$, which activates a number of parties $\mathcal{U} = \{p_1, \ldots, p_n\}$ as either honest or corrupted parties. Honest parties would faithfully follow the protocol's prescription, whereas corrupt parties are controlled by an adversary $\mathcal{A}$, which reads all their inputs/messages and sets their outputs/messages to be sent.

- A protocol's execution proceeds in rounds that model atomic time steps. At the beginning of every round, honest parties receive inputs from an environment $\mathcal{Z}$; at the end of every round, honest parties send outputs to the environment $\mathcal{Z}$.
- $\mathcal{A}$ is responsible for delivering all messages sent by parties (honest or corrupted) to all other parties. $\mathcal{A}$ cannot modify the content of messages broadcast by honest parties.
- At any point $\mathcal{Z}$ can corrupt an honest party j, which means that $\mathcal{A}$ gets access to its local state and subsequently controls party j.
- At any point of the execution, $\mathcal{Z}$ can uncorrupt a corrupted party j, which means that $\mathcal{A}$ no longer controls j. A party that becomes uncorrupt is treated in the same way as a newly spawning party, i.e., the party's internal state is re-initialised and then the party starts executing the honest protocol no longer controlled by $\mathcal{A}$.

Note that a protocol execution can be randomised, where the randomness comes from honest parties as well as from $\mathcal{A}$ and $\mathcal{Z}$. We denote by $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ the randomly sampled execution trace. More formally, view denotes the joint view of all parties (i.e., all their inputs, random coins and messages received, including those from the random oracle) in the above execution; note that this joint view fully determines the execution.

III. Editing the Blockchain

In this section we introduce an abstraction Γ of a blockchain protocol, and we describe how to extend Γ into an *editable* blockchain protocol Γ'.

A. Blockchain Protocol

We consider an immutable blockchain protocol (for instance [21]), denoted by Γ, where nodes receive inputs from the environment $\mathcal{Z}$, and interact among each other to agree on an ordered ledger that achieves persistence and liveness. The blockchain protocol Γ is characterised by a set of global parameters and by a public set of rules for validation. The protocol Γ provides the nodes with the following set of interfaces which are assumed to have complete access to the network and its users.

- $\{\mathcal{C}', \perp\} \leftarrow \Gamma.\mathsf{updateChain}$: returns a longer and *valid* chain $\mathcal{C}$ in the network (if it exists), otherwise returns $\perp$.
- $\{0, 1\} \leftarrow \Gamma.\mathsf{validateChain}(\mathcal{C})$: The chain validity check takes as input a chain $\mathcal{C}$ and returns 1 iff the chain is valid according to a public set of rules.
- $\{0, 1\} \leftarrow \Gamma.\mathsf{validateBlock}(B)$: The block validity check takes as input a block B and returns 1 iff the block is valid according to a public set of rules.
- $\Gamma.\mathsf{broadcast}(x)$: takes as input some data x and broadcasts it to all the nodes of the system.

The nodes in the Γ protocol have their own local chain $\mathcal{C}$ which is initialised with a common genesis block. The consensus in Γ guarantees the properties of persistence and liveness discussed in Section II-B.

B. Editable Blockchain

We build our editable blockchain protocol Γ' by modifying and extending the aforementioned protocol Γ. The protocol Γ' has copies of all the basic blockchain functionalities exposed by Γ through the interfaces described above, and modifies the validateChain and validateBlock algorithms in order to accommodate for edits in $\mathcal{C}$. In addition, the protocol Γ' provides the following interfaces:

- $B_j^\star \leftarrow \Gamma'.\mathsf{proposeEdit}(\mathcal{C}, j, x^\star)$: takes as input the chain $\mathcal{C}$, an index j of a block to edit and some data $x^\star$. It then returns a candidate block for B_j.
- $\{0,1\} \leftarrow \Gamma'.\mathsf{validateCand}(B_j^\star, \mathcal{C})$: takes as input a candidate block $B_j^\star$ and the chain $\mathcal{C}$ and returns 1 iff the candidate block $B_j^\star$ is valid.

The modified chain validation and block validation algorithms are presented in Algorithm 1 and Algorithm 2, respectively, while the new algorithms to propose an edit to a block and to validate candidate blocks are presented in Algorithm 3 and Algorithm 4, respectively. In Fig. 2 we formally describe the protocol Γ'.

Intuitively, we need modifications for chain validation and block validation algorithms to account for an edited block in the chain. A block that has been edited possesses a different state, that does not immediately correlate with its neighbouring blocks. Therefore, for such an edited block we need to ensure that the old state of the block (the state before the edit) is still accessible for verification.[4] We do this by storing the old state information in the block itself. This therefore requires a modified block validation algorithm and a modified chain validation algorithm overall.

We note that for simplicity our protocol is restricted to perform a single edit operation per block throughout the run of the protocol. In Appendix A we describe an extension of the protocol to accommodate for an arbitrary number of redactions per block.

Blockchain Policy. We introduce the notion of a blockchain policy $\mathcal{P}$, that determines if an edit to the chain $\mathcal{C}$ should be approved or not. The protocol Γ' is parameterised by a policy $\mathcal{P}$ that is a function that takes as input a chain $\mathcal{C}$ and a candidate block $B^\star$ (that proposes a modification to the chain $\mathcal{C}$) and it returns `accept` if the candidate block $B^\star$ complies with the policy $\mathcal{P}$, otherwise it outputs `reject`; in case the modification proposed by $B^\star$ is still being deliberated in the chain $\mathcal{C}$, then $\mathcal{P}$ returns `voting`.

In its most basic form, a policy $\mathcal{P}$ requires that a candidate block $B^\star$ should only be accepted if $B^\star$ was voted by the majority of the network within some predefined interval of blocks (or *voting period* ℓ). A formal definition follows.

Definition 1 (Policy). *A candidate block $B^\star$ generated in round r is said to satisfy the policy $\mathcal{P}$ of chain $\mathcal{C} := (B_1, \ldots, B_n)$, i.e., $\mathcal{P}(\mathcal{C}, B^\star) = \mathtt{accept}$, if it holds that $B_{r+\ell} \in \mathcal{C}^{\lceil k}$ and the ratio of blocks between B_r and $B_{r+\ell}$ containing $H(B^\star)$ (a vote for $B^\star$) is at least ρ, for $k, \ell \in \mathbb{N}$, and $0 < \rho \leq 1$, where k is the persistence parameter, ℓ is the voting period, and ρ is the ratio of votes necessary within the voting period ℓ.*

[4] Note that the protocol does *not* need to maintain the redacted data for verification, and therefore all redacted data is completely removed from the chain.

C. Protocol Description

We denote a block to be of the form $B := \langle s, x, ctr, y \rangle$, where $s \in \{0,1\}^\kappa$ is the hash of the previous block, $x \in \{0,1\}^*$ is the block data, and $y \in \{0,1\}^\kappa$ is the old state of the block data. To extend an editable chain $\mathcal{C}$ to a new longer chain $\mathcal{C}' := \mathcal{C}||B'$, the newly created block $B' := \langle s', x', ctr', y' \rangle$ sets $s' := H(ctr, G(s, x), y)$, where $\mathsf{Head}(\mathcal{C}) := \langle s, x, ctr, y \rangle$. Note that upon the creation of block B', the component y' takes the value $G(s', x')$, that represents the initial state of block B'.

During the setup of the system, the chain $\mathcal{C}$ is initialised as $\mathcal{C} := \mathsf{genesis}$, and all the users in the system maintain a local copy of the chain $\mathcal{C}$ and a pool $\mathcal{R}$ consisting candidate blocks for edits, that is initially empty. The protocol runs in a sequence of rounds r (starting with $r := 1$).

In the beginning of each round r, the users try to extend their local chain using the interface $\Gamma'.\mathsf{updateChain}$, that tries to retrieve new valid blocks from the network and append them to the local chain. Next, the users collect all the candidate blocks $B_j^\star$ from the network and validate them by using $\Gamma'.\mathsf{validateCand}$ (Algorithm 4); then, the users add all the valid candidate blocks to the pool $\mathcal{R}$. For each candidate block $B_j^\star$ in $\mathcal{R}$, the users compute $\mathcal{P}(\mathcal{C}, B_j^\star)$ to verify if the candidate block $B_j^\star$ should be adopted by the chain or not; if the output is `accept` they replace the original block B_j in the chain by the candidate block $B_j^\star$ and remove $B_j^\star$ from $\mathcal{R}$. If the output is `reject`, the users remove the candidate block $B_j^\star$ from $\mathcal{R}$, otherwise if the output is `voting` they do nothing. To create a new block B the users collect transactions from the network and store them in x; if a user wishes to endorse the edit proposed by a candidate block $B_j^\star \in \mathcal{R}$ that is still in `voting` stage, the user can vote for the candidate block $B_j^\star$ by simply adding $H(B_j^\star)$ to the data x. After the block is created and the new extended chain $\mathcal{C}' := \mathcal{C}||B$ is built, the users broadcast the new chain $\mathcal{C}'$ iff $\Gamma'.\mathsf{validateChain}(\mathcal{C}') = 1$ (Algorithm 1). Finally, if a user wishes to propose an edit to block B_j in the chain $\mathcal{C}$, she first creates the new data $x_j^\star$, that represents the modifications that she proposes to make to the data x_j, and calls proposeEdit (Algorithm 3) using the interface $\Gamma'.\mathsf{proposeEdit}$ with the chain $\mathcal{C}$, index j of the block in $\mathcal{C}$ and the new data $x_j^\star$. The algorithm returns a candidate block $B_j^\star$ that is broadcasted to the network.

Chain Validation. Given a chain $\mathcal{C}$, the user needs to validate $\mathcal{C}$ according to some set of validation rules. To do this, she uses the $\Gamma'.\mathsf{validateChain}$ interface, that is implemented by Algorithm 1. The algorithm takes as input a chain $\mathcal{C}$ and starts validating from the head of $\mathcal{C}$. In Line 5, the validity of the block B_j is checked. If the assertion in Line 6 is false and if the check in Line 7 is successful, then the block B_{j-1} is a valid edited block. In Line 7, the validity of B_{j-1} is checked in the context of a candidate block and whether the block is accepted according to the voting policy $\mathcal{P}$ of the chain.

Block Validation. To validate a block, the validateBlock algorithm (described in Algorithm 2) takes as input a block B and first validates the data included in the block according

The protocol Γ' consists of a sequence of rounds r, and is parameterised by the liveness and persistence parameters, denoted by u, k, respectively, and by a policy $\mathcal{P}$ that among other rules and constraints, determines the parameter ℓ (that is the duration of the voting period) and ρ (that is the threshold of votes within the period ℓ for a candidate block to be accepted and incorporated into the chain). A pictorial representation of the protocol can be found in Fig. 1.

Initialisation. Set the chain $\mathcal{C} \leftarrow \mathsf{genesis}$, set round $r \leftarrow 1$ and initialise an empty list of candidate blocks for edits $\mathcal{R} := \emptyset$.

For each round r of the protocol, we describe the following sequence of execution.

Chain update. At the beginning of a new round r, the nodes try to update their local chain by calling $\mathcal{C} \leftarrow \Gamma'.\mathsf{updateChain}$.

Candidate blocks pool. Collect all candidate blocks $B_j^\star$ from the network and add $B_j^\star$ to the pool of candidate blocks $\mathcal{R}$ iff $\Gamma'.\mathsf{validateCand}(\mathcal{C}, B_j^\star) = 1$; otherwise discard $B_j^\star$.

Editing the chain. For all candidate blocks $B_j^\star \in \mathcal{R}$ do:

- If $\mathcal{P}(\mathcal{C}, B_j^\star) = \texttt{accept}$, then build the new chain as $\mathcal{C} \leftarrow \mathcal{C}^{\lceil (n-j+1)} || B_j^\star ||^{j \rceil}\mathcal{C}$ and remove $B_j^\star$ from $\mathcal{R}$. For policy $\mathcal{P}$ to accept $B_j^\star$, it must be the case that the ratio of votes for $B_j^\star$ within its voting period (ℓ blocks) is at least ρ.
- If $\mathcal{P}(\mathcal{C}, B_j^\star) = \texttt{reject}$, then remove $B_j^\star$ from $\mathcal{R}$. For policy $\mathcal{P}$ to reject $B_j^\star$ it must be the case that the ratio of votes for $B_j^\star$ within its voting period (ℓ blocks) is less than ρ.
- If $\mathcal{P}(\mathcal{C}, B_j^\star) = \texttt{voting}$, then do nothing.

Creating a new block. Collects all the transaction data x from the network for the r-th round and tries to build a new block B_r by performing the following steps:

- *(Voting for candidate blocks).* For all candidate blocks $B_j^\star \in \mathcal{R}$ that the node is willing to endorse, if $\mathcal{P}(\mathcal{C}, B_j^\star) = \texttt{voting}$ then set $x \leftarrow x || H(B_j^\star)$.
- Create a new block $B := \langle s, x, ctr, G(s, x)\rangle$, such that $s = H(ctr', G(s', x'), y')$, for $\langle s', x', ctr', y'\rangle \leftarrow \mathsf{Head}(\mathcal{C})$.
- Extend its local chain $\mathcal{C} \leftarrow \mathcal{C}||B$ and iff $\Gamma'.\mathsf{validateChain}(\mathcal{C}) = 1$ then broadcast $\mathcal{C}$ to the network.

Propose an edit. The node willing to propose an edit for the block B_j, for $j \in [n]$, creates a candidate block $B_j^\star \leftarrow \Gamma'.\mathsf{proposeEdit}(\mathcal{C}, j, x^\star)$ using the new data $x^\star$, and broadcasts it to the network by calling $\Gamma'.\mathsf{broadcast}(B_j^\star)$.

Figure 2: Accountable permissionless editable blockchain protocol $\Gamma'_{\mathcal{P}}$

Algorithm 1: validateChain (implements Γ'.validateChain)

input : Chain $\mathcal{C} = (B_1, \cdots, B_n)$ of length n.
output: $\{0, 1\}$

$j := n$;
if $j = 1$ **then return** $\Gamma'.\mathsf{validateBlock}(B_1)$;
while $j \geq 2$ **do**
  $B_j := \langle s_j, x_j, ctr_j, y_j\rangle$; ▷ $B_j := \mathsf{Head}(\mathcal{C})$ when $j = n$
  if $\Gamma'.\mathsf{validateBlock}(B_j) = 0$ **then return** 0;
  if $s_j = H(ctr_{j-1}, G(s_{j-1}, x_{j-1}), y_{j-1})$ **then** $j := j - 1$;
  else if $(s_j = H(ctr_{j-1}, y_{j-1}, y_{j-1})) \wedge (\Gamma'.\mathsf{validateCand}(\mathcal{C}, B_{j-1}) = 1) \wedge (\mathcal{P}(\mathcal{C}, B_{j-1}) = \texttt{accept})$ **then** $j := j - 1$;
  else return 0;
return 1;

Algorithm 2: validateBlock (implements Γ'.validateBlock)

input : Block $B := \langle s, x, ctr, y\rangle$.
output: $\{0, 1\}$

Validate data x, if *invalid* **return** 0;
if $H(ctr, G(s, x), y) < D \vee H(ctr, y, y) < D$ **then return** 1;
else return 0;

to some pre-defined validation predicate. It then checks if the block indeed satisfies the constraints of the PoW puzzle. Apart from this check, the or ($\vee$) condition is to ensure that in case of dealing with an edited block B, the old state of B still satisfies the PoW constraints.

Proposing an Edit. Any user in the network can propose for a particular data to be removed or replaced from the blockchain. She uses the proposeEdit algorithm as described in Algorithm 3 and constructs a candidate block to replace the original block. The algorithm takes as input a chain $\mathcal{C}$, the index j of the original block and new data $x_j^\star$ that will replace the original data. If the user's intention is simply to remove all data from block B_j then $x_j^\star := \varepsilon$. It then generates a candidate block as the tuple $B_j^\star := \langle s_j, x_j^\star, ctr_j, y_j\rangle$.

Validating Candidate Blocks. When the user wishes to validate a candidate block $B_j^\star := \langle s_j, x_j^\star, ctr_j, y_j\rangle$ for the j-th block of a chain $\mathcal{C}$, she uses validateCand which is described in Algorithm 4. It retrieves the blocks B_{j-1} and B_{j+1} of index $j-1$ and $j+1$ respectively from the chain $\mathcal{C}$. In Line 5 it is checked if the link $s_j^\star$ from $B_j^\star$ to B_{j-1} holds and that the link s_{j+1} from B_{j+1} to $B_j^\star$ also satisfies the condition $s_{j+1} = H(ctr_j, y_j, y_j)$. The latter condition checks if the "old

Algorithm 3: proposeEdit (implements Γ'.proposeEdit)

input : Chain $\mathcal{C} = (B_1, \cdots, B_n)$ of length n, an index $j \in [n]$, and the new data $x_j^\star$.
output: A candidate block $B_j^\star$.

1: Parse $B_j := \langle s_j, x_j, ctr_j, y_j \rangle$;
2: Build the candidate block $B_j^\star := \langle s_j, x_j^\star, ctr_j, y_j \rangle$;
3: **return** $B_j^\star$;

link" still holds. If both checks are successful the candidate block $B_j^\star$ is considered valid, otherwise it is considered invalid.

Algorithm 4: validateCand (implements Γ'.validateCand)

input : Chain $\mathcal{C} = (B_1, \cdots, B_n)$ of length n, and a candidate block $B_j^\star$ for an edit.
output: $\{0, 1\}$

1: Parse $B_j^\star := \langle s_j, x_j^\star, ctr_j, y_j \rangle$;
2: **if** Γ'.validateBlock$(B_j^\star) = 0$ **then return** 0;
3: Parse $B_{j-1} := \langle s_{j-1}, x_{j-1}, ctr_{j-1}, y_{j-1} \rangle$;
4: Parse $B_{j+1} := \langle s_{j+1}, x_{j+1}, ctr_{j+1}, y_{j+1} \rangle$;
5: **if** $s_j^\star = H(ctr_{j-1}, y_{j-1}, y_{j-1}) \wedge s_{j+1} = H(ctr_j, y_j, y_j)$ **then return** 1;
6: **else return** 0;

IV. Security Analysis

In this section we analyse the security of our editable blockchain protocol of Fig. 2.

We assume the existence of an immutable blockchain protocol Γ, as described in Section III-A, that satisfies the properties of chain growth, chain quality and common prefix [21]. The basic intuition behind our security analysis is that, given that Γ satisfies the aforementioned properties, our editable blockchain protocol $\Gamma'_{\mathcal{P}}$, (which is Γ' parameterised by a policy $\mathcal{P}$), preserves the same properties (or a variation of the property in the case of common prefix). Therefore, our protocol behaves exactly like the immutable blockchain Γ when there are no edits in the chain, and if an edit operation was performed, it must have been approved by the policy $\mathcal{P}$. We discuss each individual property next.

Chain Growth. The chain growth property from Γ is automatically preserved in our editable blockchain Γ', since the possible edits do not allow the removal of blocks or influence the growth of the chain. We present the formal definition next, followed by a theorem stating that Γ' preserves chain growth whenever Γ satisfies chain growth.

Definition 2 (Chain Growth [21])**.** *Consider the chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of two slots sl_1, sl_2, with sl_2 at least s slots ahead of sl_1. Then it holds that* $\mathsf{len}(\mathcal{C}_2) - \mathsf{len}(\mathcal{C}_1) \geq \tau \cdot s$, *for $s \in \mathbb{N}$ and $0 < \tau \leq 1$, where τ is the speed coefficient.*

Theorem 1. *If Γ satisfies (τ, s)-chain growth, then $\Gamma'_{\mathcal{P}}$ satisfies (τ, s)-chain growth for any policy $\mathcal{P}$.*

Proof. We note that Γ' extends Γ, that by assumption satisfies chain growth. Also, note that in Γ' it is not possible to remove a block from the chain (for any policy $\mathcal{P}$), thereby reducing the length of $\mathcal{C}$. In other words, the edits performed do not alter the length of the chain. Therefore, we conclude that Γ' satisfies chain growth whenever Γ satisfies chain growth. □

Chain Quality. The chain quality property informally states that the ratio of adversarial blocks in any segment of a chain held by a honest party is no more than a fraction μ, where μ is the fraction of resources controlled by the adversary.

Definition 3 (Chain Quality [21])**.** *Consider a portion of length ℓ-blocks of a chain possessed by an honest party during any given round, for $\ell \in \mathbb{N}$. Then, the ratio of adversarial blocks in this ℓ segment of the chain is at most μ, where $0 < \mu \leq 1$ is the chain quality coefficient.*

Theorem 2. *Let H be a collision-resistant hash function. If Γ satisfies (μ, ℓ)-chain quality, then $\Gamma'_{\mathcal{P}}$ satisfies (μ, ℓ)-chain quality for any (k, ℓ, ρ)-policy where $\rho > \mu$.*

Proof. We note that the only difference in $\Gamma'_{\mathcal{P}}$ in relation to Γ is that blocks can be edited. An adversary $\mathcal{A}$ could edit an honest block B in the chain $\mathcal{C}$ into a malicious block $B^\star$ (e.g., that contains illegal content), increasing the proportion of malicious blocks in the chain, and therefore breaking the chain quality property. We show below that $\mathcal{A}$ has only a negligible probability of violating chain quality of Γ'.

Let $\mathcal{A}$ propose a malicious candidate block $B_j^\star$ for editing an honest block $B_j \in \mathcal{C}$. Since $\mathcal{A}$ possesses only μ computational power, by the chain quality property of Γ we know that the adversary mines at most μ ratio of blocks in the voting phase. As the policy stipulates, the ratio of votes has to be at least ρ for $B^\star$ to be approved, where $\rho > \mu$. Therefore, $B^\star$ can only be approved by the policy $\mathcal{P}$ if honest nodes vote for it. Observe that the adversary could try to build an "honest looking" (e.g., without illegal contents) candidate block $\widetilde{B^\star} \neq B^\star$ such that $H(\widetilde{B^\star}) = H(B^\star)$, in an attempt to deceive the honest nodes during the voting phase; the honest nodes could endorse the candidate block $B^\star$ during the voting phase, and the adversary would instead edit the chain with the malicious block $\widetilde{B^\star}$. The adversary has only a negligible chance of producing such a candidate block $B^\star$ where $H(\widetilde{B^\star}) = H(B^\star)$, since this would violate the collision-resistance property of the hash function H.

Moreover, $B^\star$ is incorporated to the chain only if it is an honest candidate block. This concludes the proof. □

Common Prefix. The common prefix property informally says that if we take the chains of two honest nodes at different time slots, the shortest chain is a prefix of the longest chain (up to the common prefix parameter k). We show the formal definition next.

Definition 4 (Common Prefix [21])**.** *The chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of the slots $sl_1 < sl_2$ are such that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$, where $\mathcal{C}_1^{\lceil k}$ denotes the chain obtained by removing the last k blocks from $\mathcal{C}_1$, where $k \in \mathbb{N}$ is the common prefix parameter.*

We remark however, that our protocol $\Gamma'_{\mathcal{P}}$ inherently does not satisfy Definition 4. To see this, consider the case where two chains $\mathcal{C}_1$ and $\mathcal{C}_2$ are held by two honest parties P_1 and P_2 at slots sl_1 and sl_2 respectively, such that $sl_1 < sl_2$. In slot r starts the voting phase (that lasts ℓ blocks) for a candidate block $B_j^\star$ proposing to edit block B_j, such that $j + k \leq r < sl_1 \leq \ell + k < sl_2$. Note that at round sl_1 the voting phase is still on, therefore $\mathcal{P}(\mathcal{C}_1, B_j^\star) = \texttt{voting}$. By round sl_2, the voting phase is complete and in case $\mathcal{P}(\mathcal{C}_2, B_j^\star) = \texttt{accept}$ the block B_j is replaced by $B_j^\star$ in $\mathcal{C}_2$. However, in chain $\mathcal{C}_1^{\lceil k}$ the j-th block is still B_j, since the edit of $B_j^\star$ is waiting to be confirmed. Therefore, $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$, thereby violating Definition 4.

The pitfall in Definition 4 is that it does not account for edits or modifications in the chain. We therefore introduce a new definition that is suited for an editable blockchain (with respect to an editing policy). The formal definition follows.

Definition 5 (Editable Common prefix)**.** *The chains $\mathcal{C}_1, \mathcal{C}_2$ of length l_1 and l_2, respectively, possessed by two honest parties at the onset of the slots $sl_1 \leq sl_2$ satisfy one of the following:*

1) $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$, or

2) for each $B_j^\star \in \mathcal{C}_2^{\lceil (l_2 - l_1)+k}$ such that $B_j^\star \notin \mathcal{C}_1^{\lceil k}$, it must be the case that $\mathcal{P}(\mathcal{C}_2, B_j^\star) = \texttt{accept}$, for $j \in [l_1 - k]$,

where $\mathcal{C}_2^{\lceil (l_2 - l_1)+k}$ denotes the chain obtained by pruning the last $(l_2 - l_1) + k$ blocks from $\mathcal{C}_2$, $\mathcal{P}$ denotes the chain policy, and $k \in \mathbb{N}$ denotes the common prefix parameter.

Intuitively, the above definition states that if there exists a block that violates the common prefix as defined in Definition 4, then it must be the case that this block is an edited block whose adoption was voted and approved according to the policy $\mathcal{P}$ in chain $\mathcal{C}_2$. We show that our protocol Γ' satisfies Definition 5 next.

Theorem 3. *Let H be a collision-resistant hash function. If Γ satisfies k-common prefix, then $\Gamma'_{\mathcal{P}}$ satisfies k-editable common prefix for a (k, ℓ, ρ)-policy.*

Proof. If no edits were performed in a chain $\mathcal{C}$, then the protocol $\Gamma'_{\mathcal{P}}$ behaves exactly like the immutable protocol Γ, and henceforth the common prefix property follows directly.

However, in case of an edit, consider an adversary $\mathcal{A}$ that proposes a candidate block $B_j^\star$ to edit B_j in chain $\mathcal{C}_2$, which is later edited by an honest party P_2 at slot sl_2. Observe that by the collision resistance property of H, $\mathcal{A}$ is not able to efficiently produce another candidate block $\widetilde{B_j^\star} \neq B_j^\star$ such that $H(\widetilde{B_j^\star}) = H(B_j^\star)$. Therefore, since P_2 is honest and adopted the edit $B_j^\star$ in $\mathcal{C}_2$, it must be the case that $B_j^\star$ received enough votes such that $\mathcal{P}(\mathcal{C}_2, B_j^\star) = \texttt{accept}$. This concludes the proof. $\square$

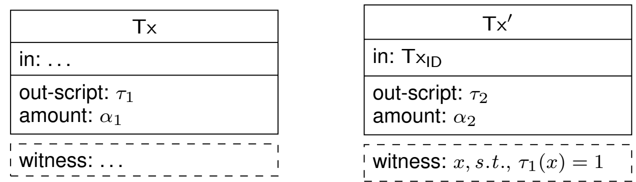


Figure 3: The structure of a transaction in Bitcoin. The transaction Tx' is spending the output τ_1 of transaction Tx.

How the properties play together: By showing that Γ' satisfies the three aforementioned properties, we show that $\Gamma'_{\mathcal{P}}$ is a live and persistent blockchain protocol immutable against edits not authorised by the policy $\mathcal{P}$.

The editable common prefix property ensures that only policy approved edits are performed on the chain. The Chain quality property, for a (k, ℓ, ρ)-policy $\mathcal{P}$ where $\rho > \mu$, ensures that an adversary does not get a disproportionate contribution of blocks to the chain.

V. Integrating into Bitcoin

In this section we describe how our generic editable blockchain protocol (Fig. 2) can be integrated into Bitcoin. For simplicity, we consider one redaction per block and the redaction is performed on one or more transactions included in the block. The extension of the generic protocol for multiple redactions (described in Appendix A) can be immediately applied to the construction described in this section. Next, we give a brief background on the Bitcoin protocol.

A. Bitcoin Basics

Transactions. A simple transaction Tx in Bitcoin has the following basic structure: an input script, an output script with a corresponding amount, and a witness. More complex transactions may have multiple input and output scripts and/or more complex scripts. A transaction Tx' that spends some output τ of Tx, has the ID of Tx in its input, denoted by $\mathsf{Tx}_{\mathsf{ID}} := H(\mathsf{Tx})$, and a witness x that satisfies the output script τ of Tx (as shown in Fig. 3). The amount α_2 being spent by the output script τ_2 needs to be smaller (or equal) than the amount α_1 of τ_1. The most common output scripts in Bitcoin consists of a public key, and the witness x is a signature of the transaction computed using the corresponding secret key. We refer the reader to [1] for a comprehensive overview of the Bitcoin scripting language.

Insertion of Data. Users are allowed to propose new transactions containing arbitrary data, that are then sent to the Bitcoin network for a small fee. Data can be inserted into specific parts of a Bitcoin transaction, namely the output script, input script and witness. Matzutt et al. [34] provide a quantitative analysis of data insertion methods in Bitcoin. According to their analysis, `OP_RETURN` and coinbase transactions are the major pockets apart from some non-standard transactions, where data is inserted.

Block Structure. A Bitcoin block consists of two parts, namely the block header, and a list of all transactions within the block. The structure of the block header is detailed in Fig. 5, whereas a pictorial representation of the list of transactions can be found in Fig. 6.

B. Modifying the Bitcoin Protocol

In this section we detail the modifications to the Bitcoin protocol necessary to integrate it to our generic editable blockchain protocol of Section III. The resulting protocol is a version of Bitcoin that allows for redaction of (harmful) data from its transactions.

By redaction of transactions, we mean removing data from a transaction without making other changes to the remaining components of the transaction. As shown in Fig. 4, consider a transaction Tx_1 that contains some harmful data in its output script, and let $\mathsf{Tx}_1^\star$ be a candidate transaction to replace Tx in the chain, where $\mathsf{Tx}_1^\star$ is exactly the same as Tx_1, except that the harmful data is removed.

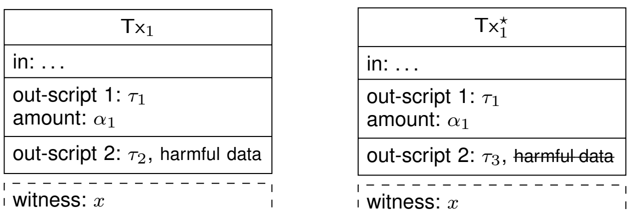


Figure 4: The transaction Tx_1 on the left contains harmful data, and the candidate transaction $\mathsf{Tx}_1^\star$ on the right contains a copy of all the fields of Tx_1, with exception of the harmful data.

Proposing Redactions. A user who wishes to propose a redaction proceeds as follows: First, constructs a special transaction editTx (as shown in Fig. 7) containing $\mathsf{Tx}_{1_{\mathsf{ID}}}$ and $\mathsf{Tx}^\star_{1_{\mathsf{ID}}}$, that respectively denotes the hash of the transaction Tx_1 being redacted, and the hash of $\mathsf{Tx}_1^\star$ that is the candidate transaction to replace Tx_1 in the chain[5]. Then, broadcasts the special transaction editTx and the candidate transaction $\mathsf{Tx}_1^\star$ to the network; editTx requires a transaction fee to be included in the blockchain, while $\mathsf{Tx}_1^\star$ is added to a pool of candidate transactions[6]. The candidate transaction $\mathsf{Tx}_1^\star$ is validated by checking its contents with respect to Tx_1, and if it is valid, then it can be considered for voting.

Redaction Policy. The redactable Bitcoin protocol is parameterised by a policy parameter $\mathcal{P}$ (Definition 1). The policy $\mathcal{P}$ dictates the requirements and constraints for redaction operations in the blockchain. An informal description of a (basic) policy for Bitcoin would be:

A proposed redaction is approved valid if the following conditions hold:

- It is identical to the transaction being replaced, except that it can remove data.
- It can only remove data that can never be spent, e.g., `OP_RETURN` output scripts.
- It does not redact votes for other redactions in the chain.
- It received more than 50% of votes in the 1024 consecutive blocks (voting period) after the corresponding editTx is stable in the chain.

where voting for a candidate transaction $\mathsf{Tx}_1^\star$ simply means that the miner includes $\mathsf{editTx}_{\mathsf{ID}} = H(\mathsf{Tx}_{1_{\mathsf{ID}}} || \mathsf{Tx}^\star_{1_{\mathsf{ID}}})$ in the coinbase (transaction) of the new block he produces. After the voting phase is over, the candidate transaction is removed from the candidate pool.

The reason for restricting the redactions to non-spendable components of a transaction (e.g., `OP_RETURN`) is that, permitting redactions on spendable content could lead to potential misuse (Section VII) and future inconsistencies within the chain. We stress however, that this is not a technical limitation of our solution, but rather a mechanism to remove the burden of the user on deciding what redactions could cause inconsistencies on the chain in the future. We feel that the aforementioned policy is suitable for Bitcoin, but as policies are highly dependent on the application, a different policy can be better suited for different settings.

Value	Description
`hash_prev`	hash of the previous block header
`merkle_root`	root of the merkle tree (whose the leaves are the transactions)
`difficulty`	the difficulty of the proof-of-work
`timestamp`	the timestamp of the block
`nonce`	nonce used in proof-of-work
`old_merkle_root`	root of the merkle tree of old set of transactions

Figure 5: Structure of the Bitcoin block header. The last highlighted field (`old_merke_root`) is only included in the block header of the extended (editable) protocol.

New Block Structure. To account for redactions, the block header must accommodate an additional field called $\mathsf{old\_merkle\_root}$. When a block is initially created, i.e., prior to any redaction, this new field takes the same value as $\mathsf{merkle\_root}$. For a redaction request on block B_j, that proposes to replace Tx_1 with the candidate transaction $\mathsf{Tx}_1^\star$, the transactions list of the candidate block $B_j^\star$ (that will replace B_j) must contain $\mathsf{Tx}_{1_{\mathsf{ID}}} = H(\mathsf{Tx}_1)$ in addition to the remaining transactions. A new $\mathsf{merkle\_root}$ is computed for the new set of transactions, while $\mathsf{old\_merkle\_root}$ remains unchanged. To draw parallels with the abstraction we described in Section III-A, $G(s, x)$ is analogous to $\mathsf{merkle\_root}$ and y is analogous to $\mathsf{old\_merkle\_root}$.

Block Validation. The validation of a block consists of the steps described below.

- *Validating transactions*: The block validates all the transactions contained in its transactions list; the validation of non-redacted transactions is performed in the same way as in the immutable version of the protocol. Trans-

[5]We note that our transaction ID is Segwit compatible, as the witness is not used with the hash H to generate a transaction's ID.

[6]If a candidate transaction does not have a corresponding editTx in the blockchain then the transaction is not included in the candidate pool, and it is treated as spam instead.

Tx_1
Tx_2
Tx_3
$\vdots$

(a) Non-redacted.

$\mathsf{Tx}_1^\star, \mathsf{Tx}_{1_{\mathsf{ID}}}$
Tx_2
Tx_3
$\vdots$

(b) Redacted transaction Tx_1.

Figure 6: List of transactions contained within a block before (left) and after (right) redacting a transaction in the block.

editTx
in: ...
out-script: $\mathsf{Tx}_{1_{\mathsf{ID}}}, \mathsf{Tx}^\star_{1_{\mathsf{ID}}}$
witness: ...

Figure 7: The special transaction editTx is broadcasted to the network to propose a redaction of transaction Tx_1 for the candidate transaction $\mathsf{Tx}_1^\star$.

actions that have been previously redacted require a special validation that we describe next. Consider the case presented in Fig. 4, where Tx_1 is replaced by $\mathsf{Tx}_1^\star$. The witness x was generated with respect to $\mathsf{Tx}_{1_{\mathsf{ID}}}$ and is not valid with respect to $\mathsf{Tx}^\star_{1_{\mathsf{ID}}}$. Fortunately, the old state $\mathsf{Tx}_{1_{\mathsf{ID}}}$ (hash of the redacted transaction) is stored, as shown in Fig. 6b, ensuring that the witness x can be successfully validated with respect to the old version of the transaction. Therefore, we can ensure that all the transactions included in the block have a valid witness, or in case of redacted transactions, the old version of the transaction had a valid witness. To verify that the redaction was approved in the chain one needs to find a corresponding editTx (Fig. 7) in the chain, and verify that it satisfies the chain's policy.
- *PoW verification*: The procedure to verify the PoW puzzle is described in Algorithm 2. If the block contains an edited transaction, i.e., $\mathsf{old\_merkle\_root} \neq \mathsf{merkle\_root}$, then substitute the value in $\mathsf{hash\_merkle\_root}$ with that in $\mathsf{old\_merkle\_root}$ and check if the hash of this new header is within T.

Chain Validation. To validate a full chain a miner needs to validate all the blocks within the chain. The miner can detect if a block has been redacted by verifying its hash link with the next block; in case of a redacted block, the miner verifies if the redaction was approved according to the chain's policy. The miner rejects a chain as invalid if *any* of the following holds: (1) a block's redaction was not approved according to the policy, (2) the $\mathsf{merkle\_root}$ value of the redacted block is incorrect with respect to the set of transactions (that contains the hash of the redacted transaction) or (3) a previously approved redaction was not performed on the chain.

Transaction Consistency. Removing a transaction entirely or changing spendable data of a transaction may result in serious inconsistencies in the chain. For example, consider a transaction Tx_1 that has two outputs denoted by A and B, where the second output B has a data entry and the first output A contains a valid spendable script that will be eventually spent by some other transaction Tx'. If the redaction operation performed on Tx_1 affects the output script of A, Tx' may become invalid, causing other transactions to become invalid. A similar problem may arise if the redaction is performed on the input part of Tx_1 enabling the user who generated Tx_1 to possibly double spend the funds. Therefore, we only allow redactions that do not affect a transaction's consistency with past and future events.

Redaction and Retrievability. The redaction policy $\mathcal{P}$ for Bitcoin restricts redactions to only those operations that do not violate a transaction's consistency. This means that we do not allow monetary transactions to be edited (such as standard coin transfer). We stress, however that the main objective of redacting a transaction Tx is to prevent some malicious content x, that is stored inside Tx, from being broadcasted as part of the chain, thereby ensuring that the chain and its users are legally compliant. Note that we cannot prevent an adversary from locally storing and retrieving the data x, even after its redaction, since the content was publicly stored in the blockchain. In this case, the user that willingly keeps the malicious (and potentially illegal) data x will be liable.

Accountability. Our proposal offers accountability during and after the voting phase is over. Moreover, the accountability during the voting phase prevents the problem of transaction inconsistencies discussed above.

- *Voting Phase Accountability*: During the voting phase, anyone can verify all the details of a redaction request. The old transaction and the proposed modification (via the candidate transaction) are up for public scrutiny. It is publicly observable if a miner misbehaves by voting for a redaction request that, apart from removing data, also tampers with the input or (a spendable) output of the transaction, in turn affecting its transaction consistency. This could discourage users from using the system due to its unreliability as a public ledger for monetary purposes. Since the miners are heavily invested in the system and are expected to behave rationally, they would not vote for such an edit request (that is against the policy) during the voting phase.
- *Victim Accountability*: After a redaction is performed, our protocol allows the data owner, whose data was removed, to claim that it was indeed her data that was removed. Since we store the hash of the old transaction along with the candidate transaction in the edited block (refer to Fig. 6b), it is possible for a user that possesses the old data (that was removed) to verify it against the hash that is stored in the redacted block. This enforces accountability on the miners of the network who vote for a redaction request by discouraging them from removing benign data. At the same time, our protocol guarantees protection against false claims, as the hash verification would fail.

VI. Proof-of-Concept Implementation

In this section we report on a Python proof-of-concept implementation used for evaluating our approach. We implement a full-fledged Blockchain system based on Python 3 that mimics all the basic functionalities of Bitcoin. Specifically, we include a subset of Bitcoin's script language that allows us to insert arbitrary data into the chain, which can be redacted afterwards. The redacting mechanism is built upon the proposed modifications to Bitcoin that we describe in Section V. For conceptual simplicity we rely on PoW as the consensus mechanism.

A. Benchmarking

We detail the performance achieved by our implementation running several experiments. The benchmarking was performed in a virtual environment on a Linux server with the following specifications.

- Intel Xeon Gold 6132 CPU @ 2.60GHz
- 128GB of RAM
- Debian Linux 4.9.0-6-amd64
- Python 3.5.3.

We measure the run time of Algorithm 1 by validating chains of varying lengths (i.e., number of blocks) and with different numbers of redactions in the chain. For each experiment, a new chain is created and validated 50 times, then the arithmetic mean of the run time is taken over all runs. Each chain consists of up to $50,000$ blocks, where each block contains $1,000$ transactions. Note that a chain of size $50,000$ blocks approximates a one year snapshot of the bitcoin blockchain.

The great variation of the results shown in the experiments is due to the randomness involved in the chain creation and validation process, since each chain will contain its own set of (different) transactions, slightly influencing the run time.

Overhead Compared to Immutable Blockchain. For the first series of experiments, we generate chains of length ranging from $10,000$ up to $50,000$ blocks. We generate both, immutable and redactable chains (with no redactions). The goal here is to measure the overhead that comes with the integration of our redactable blockchain protocol with an immutable blockchain when there are no redactions performed. The results in Fig. 8 indicate that there is only a tiny overhead. Interestingly, we note that as the size of the chain grows, the overhead tends to get smaller; this is because on a chain without redactions the only extra step required is to check if there are any votes in the coinbase transaction of a new block, what becomes negligible compared to the verification time as the chain grows larger.

Overhead by Number of Redactions. For the second series of experiments, we generate redactable chains with the number of redactions ranging from 2% $(1,000$ redactions) to 10% $(5,000$ redactions) of the blocks. The redacted transactions within a block contains dummy data consisting of 4 bytes that are removed during the experiment. This experiment is intended to measure the overhead with respect to the number of redactions performed in a chain compared to a redactable chain *with no* redactions. The results in Fig. 9 show that the overhead tends to be at most linear in the number of redactions, since in our prototype instead of looking ahead whether there is a redaction request and a sufficient number of votes, we keep track of the redaction request and wait for its votes and eventual confirmation.

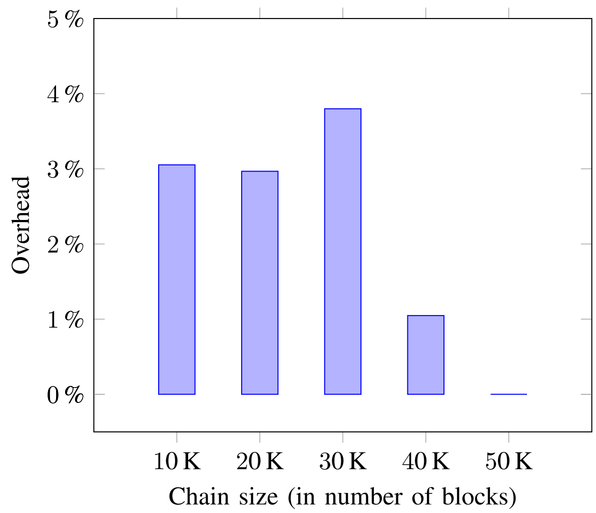


Figure 8: The graph shows the validation time overhead required to validate a redactable chain (with no redactions) compared to an immutable chain.

Overhead by the Voting Parameters ℓ and ρ. In the last series of experiments, we consider chains with 1% of the blocks redacted. We vary the voting period ℓ to measure how it influences the validation time compared to a chain with 1% of blocks redacted but with a voting period of $\ell = 5$. The threshold of votes ρ is set to $\left(\lfloor \frac{\ell}{2} \rfloor + 1\right)/\ell$ (i.e., requiring majority number of blocks in the voting period to contain votes for approving a redaction). The results in Fig. 10 show that the overhead is very small (even negligible for small sizes of ℓ) and tends to be at most linear in ℓ. This meets our expectations, since the overhead in validation time originates from keeping and increasing the voting counts over the voting period ℓ. In the worst case, where $\rho = 1$ we need to keep track of the voting count over the entire voting period.

VII. Discussion

In this section we discuss some of the generic attacks on our system and how it is immune to such attacks.

Unapproved Editing. A malicious miner could pass off an edit on the blockchain that does not satisfy the network's policy. This can occur if the miner presents the blockchain with an edit that has not been considered for voting, or has gathered insufficient votes. In any of the above cases, it is possible for any user in the network to account for an edit by verifying in the chain if the exact edit presented by the miner is approved or not. And since majority of the miners in

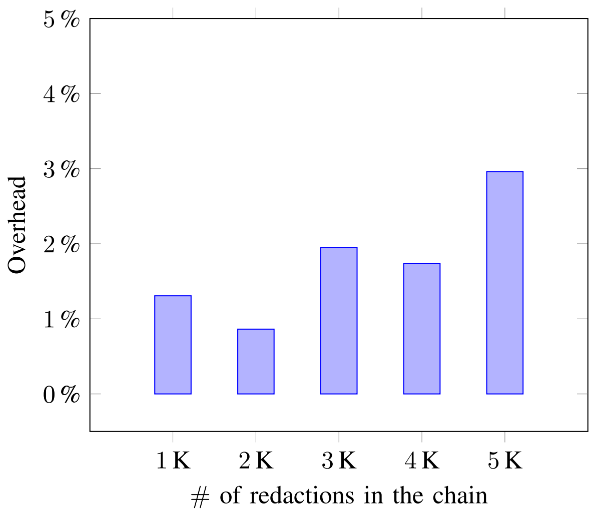


Figure 9: The graph shows the validation time overhead required to validate a chain for an increasing number of redactions, compared to a redactable chain with no redactions.

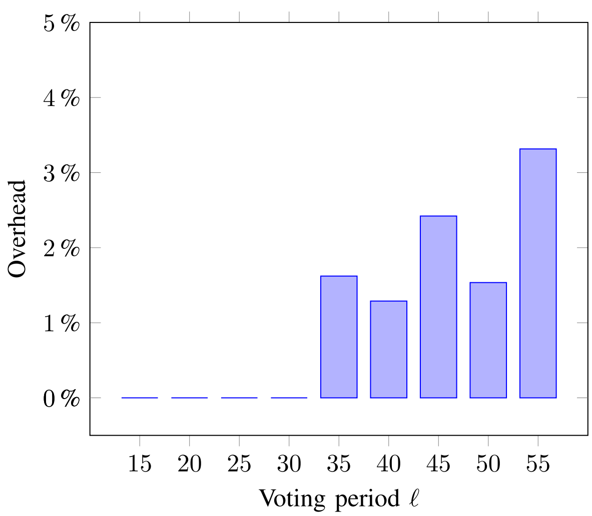


Figure 10: The graph shows the validation time overhead required to validate a chain (with 1% of the blocks redacted) for increasing voting periods, compared to a chain (with 1% of the blocks redacted) on a fixed voting period of $\ell = 5$.

the network is honest, the user accepts an approved edit as an honest edit.

Scrutiny of Candidate Blocks. It is in the interest of the (honest) miners and the system as a whole, to actively scrutinise a candidate block and decide on voting based on its merit. Therefore, the miners are strongly discouraged from using a default strategy in voting, e.g., always vote for a candidate block without scrutiny, using a pre-determined strategy that is agnostic to what the candidate block is proposing.

Denial of Service. A malicious miner may try to flood the network with edit requests as an attempt to slow down transaction confirmation in the chain. However, the miner is deterred from doing this because he incurs the cost of a transaction fee for the editTx that is part of his edit request similar to other standard transactions. Moreover, it may also be the case for the editTx to incur a higher transaction fee as a strong deterrent against spamming.

False Victim. A malicious user may wrongly claim that a particular transaction related to him was edited. For example, he may claim that some monetary information was changed where he was the beneficiary. Since such an edit could affect the trust in the system, the user could potentially affect the credibility of the entire system. We prevent such an attack through victim accountability of our protocol. We can verify the user's claim against the hash of the old version of the transaction that is stored in the chain itself. Given the hash function is collision resistant, a wrong claim would fail the check.

Double Spend Attacks. Consider a scenario where a malicious user is the recipient of a transaction. If this transaction was edited by removing some data stored in it, the hash of the new version of the transaction is different. If the miner had already spent the funds from the old version of this transaction, after the edit, he may attempt a double spend by exploiting the new version of the transaction. This is prevented by associating the new version and the old version of the edited transaction with each other, thereby noticing such a double spend. If the funds had already been spent, the old version would be a spent transaction. Because the edit that is performed does not conflict with the consistency of the transaction, the new version of the transaction would also be a spent transaction.

Consensus delays. Consider a scenario where two different users hold chains with a different set of redacted blocks, and therefore cannot arrive at a consensus on the final state of the chain, what may result in delays. Assuming the miners have not locally redacted blocks on their own and have behaved honestly according to the protocol, this scenario would mean that the different set of redacted blocks in the chains held by the two miners have been approved by the policy. However, this would be a blatant violation of the Editable common prefix property of our protocol (Theorem 3).

VIII. Related Work

1) Bitcoin and Applications: Several works [8], [12], [40] have analysed the properties and extended the features of the Bitcoin protocol. Bitcoin as a public bulletin board has found several innovative applications far beyond its initial scope, e.g., to achieve fairness and correctness in secure multi-party computation [9], [7], [15], [31], to build smart contracts [30], to distributed cryptography [6], and more [32], [29], [16].

2) Content Insertion in Bitcoin: There have been several works [13], [35], [36], [42], [45], [46] on analysing and assessing the consequences of content insertions in public blockchains. They shed light on the distribution and the usage of such inserted data entries. The most recent work of Matzutt et al. [34] gives a comprehensive quantitative analysis of illicit content insertions including, insertion techniques, potential risks and rational incentives. They also show that compared to other attacks [20], [24] on Bitcoin system, illicit content insertion can pose immediate risks to all users of the system.

3) Proactive Countermeasures: Proactive measures to detect illicit material circulated in the network and detecting them have been studied [44], [27], [25]. In a blockchain setting, preventive solutions [19], [17], [37] focus on maintaining only monetary information instead of the entire ledger history. Matzutt et al. [33] use a rational approach of discouraging miners from inserting harmful content into the blockchain. They advocate a minimum transaction fee and mitigation of transaction manipulatability as a deterrent for the same.

ACKNOWLEDGMENT

This work is a result of the collaborative research project PROMISE (16KIS0763) by the German Federal Ministry of Education and Research (BMBF). FAU authors were also supported by the German research foundation (DFG) through the collaborative research center 1223, and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). NCT is a research cooperation between the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Technische Hochschule Nürnberg Georg Simon Ohm (THN).

REFERENCES

[1] "Bitcoin script," https://en.bitcoin.it/wiki/Script.

[2] "Interpol cyber research identifies malware threat to virtual currencies," 2015, https://tinyurl.com/y9wfekr6.

[3] "Accenture files patent for editable blockchain," 2016, https://tinyurl.com/yblq9zdp.

[4] "Downside of bitcoin: A ledger that can't be corrected," 2016, https://tinyurl.com/ydxjlf9e.

[5] "Child porn on bitcoin? why this doesn't mean what you might think," 2018, https://tinyurl.com/y8yo8ml3.

[6] M. Andrychowicz and S. Dziembowski, "PoW-based distributed cryptography with no trusted setup," in *CRYPTO 2015, Part II*, ser. LNCS, R. Gennaro and M. J. B. Robshaw, Eds., vol. 9216. Springer, Heidelberg, Aug. 2015, pp. 379–399.

[7] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Fair two-party computations via bitcoin deposits," in *FC 2014 Workshops*, ser. LNCS, R. Böhme, M. Brenner, T. Moore, and M. Smith, Eds., vol. 8438. Springer, Heidelberg, Mar. 2014, pp. 105–121.

[8] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek, "Modeling bitcoin contracts by timed automata," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2014, pp. 7–22.

[9] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 443–458.

[10] G. Ateniese, M. T. Chiaramonte, D. Treat, B. Magri, and D. Venturi, "Multiple-link blockchain," Oct. 10 2017, uS Patent 9,785,369.

[11] ——, "Rewritable blockchain," May 8 2018, uS Patent 9,967,096.

[12] G. Ateniese, A. Faonio, B. Magri, and B. De Medeiros, "Certified bitcoins," in *International Conference on Applied Cryptography and Network Security*. Springer, 2014, pp. 80–96.

[13] 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.

[14] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, pp. 913–930.

[15] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *CRYPTO 2014, Part II*, ser. LNCS, J. A. Garay and R. Gennaro, Eds., vol. 8617. Springer, Heidelberg, Aug. 2014, pp. 421–439.

[16] L. Breidenbach, I. Cornell Tech, P. Daian, F. Tramer, and A. Juels, "Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018.

[17] J. Bruce, "The mini-blockchain scheme," *White paper*, 2014.

[18] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors," in *IACR International Workshop on Public Key Cryptography*. Springer, 2017, pp. 152–182.

[19] A. Chepurnoy, M. Larangeira, and A. Ojiganov, "Rollerchain, a blockchain with safely pruneable full blocks," *arXiv preprint arXiv:1603.07926*, 2016.

[20] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *FC 2014*, ser. LNCS, N. Christin and R. Safavi-Naini, Eds., vol. 8437. Springer, Heidelberg, Mar. 2014, pp. 436–454.

[21] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *EUROCRYPT 2015, Part II*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, Heidelberg, Apr. 2015, pp. 281–310.

[22] B. Gipp, J. Kosti, and C. Breitinger, "Securing video integrity using decentralized trusted timestamping on the bitcoin blockchain." in *MCIS*, 2016, p. 51.

[23] B. Gipp, N. Meuschke, and A. Gernandt, "Decentralized trusted timestamping using the crypto currency bitcoin," *arXiv preprint arXiv:1502.04015*, 2015.

[24] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network." in *USENIX Security Symposium*, 2015, pp. 129–144.

[25] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: building robust firewalls for software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 97–102.

[26] L.-D. Ibanez, K. O'Hara, and E. Simperl, "On blockchains and the general data protection regulation," 2018.

[27] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, "Implementing a distributed firewall," in *ACM CCS 00*, S. Jajodia and P. Samarati, Eds. ACM Press, Nov. 2000, pp. 190–199.

[28] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[29] A. Kiayias and Q. Tang, "Traitor deterring schemes: Using bitcoin as collateral for digital content," in *ACM CCS 15*, I. Ray, N. Li, and C. Kruegel:, Eds. ACM Press, Oct. 2015, pp. 231–242.

[30] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 839–858.

[31] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in *ACM CCS 14*, G.-J. Ahn, M. Yung, and N. Li, Eds. ACM Press, Nov. 2014, pp. 30–41.

[32] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *ACM CCS 15*, I. Ray, N. Li, and C. Kruegel:, Eds. ACM Press, Oct. 2015, pp. 195–206.

[33] 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.

[34] 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.

[35] R. Matzutt, O. Hohlfeld, M. Henze, R. Rawiel, J. H. Ziegeldorf, and K. Wehrle, "Poster: I don't want that content! on the risks of exploiting bitcoin's blockchain as a content store," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM, 2016, pp. 1769–1771.

[36] 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.

[37] A. Molina and H. Schoenfeld, "Pascalcoin version 2," *White paper*, 2017.

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

[39] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, 2017, pp. 643–673.

[40] R. Pass and A. Shelat, "Micropayments for decentralized currencies," in *ACM CCS 15*, I. Ray, N. Li, and C. Kruegel:, Eds. ACM Press, Oct. 2015, pp. 207–218.

[41] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017, pp. 315–324.

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

[43] I. Ray, N. Li, and C. Kruegel:, Eds., *ACM CCS 15*. ACM Press, Oct. 2015.

[44] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[45] K. Shirriff, "Hidden surprises in the bitcoin blockchain and how they are stored: Nelson mandela, wikileaks, photos, and python software," *Ken Shirriff's blog (accessed July 2017) http://www. righto. com/2014/02/ascii-bernanke-wikileaks-photographs. html*, 2014.

[46] M. D. Sleiman, A. P. Lauf, and R. Yampolskiy, "Bitcoin message: Data insertion on a proof-of-work cryptocurrency system," in *Cyberworlds (CW), 2015 International Conference on*. IEEE, 2015, pp. 332–336.

[47] G. Tziakouris, "Cryptocurrencies—a forensic challenge or opportunity for law enforcement? an interpol perspective," *IEEE Security & Privacy*, vol. 16, no. 4, pp. 92–94, 2018.

APPENDIX

A. Protocol extension for multiple redactions

In this section we sketch an extension to the protocol of Fig. 2 to accommodate multiple redactions per block.

The intuition behind the extension is simple enough to be explained in this paragraph; a block can potentially be redacted n times and each redaction $B_j^\star$ of the block B_j that is approved *must* contain information about the entire history of previous redactions. In our extension, this information is stored in the $y_j^\star$ component of the candidate block $B_j^\star$. We now sketch the required protocol changes.

Proposing an Edit. To propose a redaction for block $B_j := \langle s_j, x_j, ctr_j, y_j \rangle$ the user must build a candidate block $B_j^\star$ of the following form: $B_j^\star := \langle s_j, x_j^\star, ctr_j, y_j^\star \rangle$, where $y_j^\star := y_j || G(s_j, x_j)$ iff $y_j \neq G(s_j, x_j)$. Note that for the first redaction of B_j, we have that $y_j = G(s_j, x_j)$, and therefore $y_j^\star := G(s_j, x_j)$.

Block Validation. To validate a block, the users run the validateBlockExt algorithm described in Algorithm 5. Intuitively, the algorithm performs the same operations as Algorithm 2, except that it takes into account the possibility of the block being redacted multiple times. Observe that by parsing y as $y^{(1)}||y^{(2)}||...||y^{(l)}$, we are considering a block that has been redacted a total of l times and $y^{(1)}$ denotes the original state information of the unredacted version of the block.

Voting for Candidate Blocks. To vote for a redaction, we additionally define the following interface.

- $H(ctr, G(s, x^\star), y^\star) \leftarrow \Gamma'.\mathsf{Vt}(B^\star)$: takes as input a candidate block $B^\star$ and parses $B^\star$ as $(s, x^\star, ctr, y^\star)$. It outputs the hash value $H(ctr, G(s, x^\star), y^\star)$ as a vote for the candidate block $B^\star$.

Algorithm 5: validateBlockExt

input : Block $B := \langle s, x, ctr, y \rangle$.
output: $\{0, 1\}$

Validate data x, if *invalid* **return** 0;
Parse y as $y^{(1)}||y^{(2)}||...||y^{(l)}$, where $y_j^{(i)} \in \{0,1\}^\kappa\ \forall i \in [l]$;
if $(H(ctr, G(s, x), y) < D) \vee (H(ctr, y^{(1)}, y^{(1)}) < D)$
 then return 1;
else return 0;

Algorithm 6: validateCandExt

input : Chain $\mathcal{C} = (B_1, \cdots, B_n)$ of length n, and a candidate block $B_j^\star$ for an edit.
output: $\{0, 1\}$

Parse $B_j^\star := \langle s_j, x_j^\star, ctr_j, y_j \rangle$;
Parse y_j as $y_j^{(1)}||y_j^{(2)}||...||y_j^{(l)}$, where $y_j^{(i)} \in \{0,1\}^\kappa\ \forall i \in [l]$;
if $\Gamma'.\mathsf{validateBlockExt}(B_j^\star) = 0$ **then return** 0;
Parse $B_{j-1} := \langle s_{j-1}, x_{j-1}, ctr_{j-1}, y_{j-1} \rangle$;
Parse y_{j-1} as $y_{j-1}^{(1)}||y_{j-1}^{(2)}||...||y_{j-1}^{(l')}$, where $y_{j-1}^{(i)} \in \{0,1\}^\kappa$ $\forall i \in [l']$;
Parse $B_{j+1} := \langle s_{j+1}, x_{j+1}, ctr_{j+1}, y_{j+1} \rangle$;
if $s_j \neq H(ctr_{j-1}, y_{j-1}^{(1)}, y_{j-1}^{(1)}) \vee s_{j+1} \neq H(ctr_j, y_j^{(1)}, y_j^{(1)})$ **then return** 0;
for $i \in \{2, \ldots, n\}$ **do**
  if *the fraction of votes for* $H(ctr, y_j^{(i)}, y_j^{(1)}|| \ldots ||y_j^{(i-1)})$ *in the chain* $\mathcal{C}$ *is* not *at least* ρ *within its voting period of* ℓ *blocks* **then return** 0;
return 1

The voting interface is invoked by users that wish to endorse a candidate block by including a vote in the newly mined block (if the candidate block is still in its voting phase). Accordingly the policy $\mathcal{P}$ of the chain for redactions checks if a candidate block has received at least a ratio of ρ votes (as output by the $\Gamma'.\mathsf{Vt}$) in a span of ℓ blocks after immediately its proposal.

Candidate Block Validation. If a block B_j is being redacted more than once, then the corresponding candidate block $B_j^\star$

Algorithm 7: validateChainExt

input : Chain $\mathcal{C} = (B_1, \cdots, B_n)$ of length n.
output: $\{0, 1\}$

$j := n$;
if $j = 1$ **then return** $\Gamma'.\mathsf{validateBlockExt}(B_1)$;
while $j \geq 2$ **do**
  $B_j := \langle s_j, x_j, ctr_j, y_j \rangle$; ▷ $B_j := \mathsf{Head}(\mathcal{C})$ when $j = n$
  $B_{j-1:} := \langle s_{j-1}, x_{j-1}, ctr_{j-1}, y_{j-1} \rangle$;
  Parse y_j as $y_j^{(1)}|| \ldots ||y_j^{(l)}$, where $y_j^{(i)} \in \{0,1\}^\kappa\ \forall i \in [l]$;
  Parse y_{j-1} as $y_{j-1}^{(1)}|| \ldots ||y_{j-1}^{(l')}$, where $y_{j-1}^{(i)} \in \{0,1\}^\kappa$ $\forall i \in [l']$;
  if $\Gamma'.\mathsf{validateBlockExt}(B_j) = 0$ **then return** 0;
  if $s_j = H(ctr_{j-1}, G(s_{j-1}, x_{j-1}), y_{j-1})$ **then** $j := j - 1$;
  else if $s_j = H(ctr_{j-1}, y_{j-1}^{(1)}, y_{j-1}^{(1)}) \wedge$ $\Gamma'.\mathsf{validateCandExt}(\mathcal{C}, B_{j-1}) = 1 \wedge \mathcal{P}(\mathcal{C}, B_{j-1}) = \mathtt{accept}$ **then** $j := j - 1$;
  else return 0;
return 1;

needs to be validated for accounting for the multiple redactions that happened before; for each redaction of B_j, the votes for that redaction must exist in the chain $\mathcal{C}$. $\mathsf{validateCandExt}$ (described in Algorithm 6) validates such a candidate block.

Chain Validation. To validate a chain, the user runs the $\mathsf{validateChainExt}$ algorithm (described in Algorithm 7). The only change compared to Algorithm 1 is that now y_j is parsed as $y_j^{(1)}||\dots||y_j^{(l)}$ where the initial unredacted state of the block is stored in $y^{(1)}$.