Redactable Blockchain Protocols with Instant Redaction

Jing Xu
xujing@iscas.ac.cn
Institute of Software, Chinese Academy of Sciences

Xinyu Li
xinyuli1920@gmail.com
Institute of Software, Chinese Academy of Sciences
&&The University of Hong Kong

Lingyuan Yin
lingyuan2018@iscas.ac.cn
Institute of Software, Chinese Academy of Sciences

Yuan Lu
luyuan@iscas.ac.cn
Institute of Software, Chinese Academy of Sciences

Qiang Tang
qiang.tang@sydney.edu.au
The University of Sydney

Zhenfeng Zhang
zhenfeng@iscas.ac.cn
Institute of Software, Chinese Academy of Sciences

ABSTRACT

Blockchain technologies have received a great amount of attention, and its immutability is paramount to facilitate certain applications requiring persistent records. However, in many other use-cases, tremendous real-world incidents have exposed the harm of strict immutability. For example, illicit data stored in immutable blockchain poses numerous challenges for law enforcement agencies such as Interpol, and millions of dollars are lost due to the vulnerabilities of immutable smart contract. Moreover, "Right to be Forgotten" (a.k.a. data erasure) has been imposed in new European Union's General Data Protection Regulation, thus causing immutable blockchains no longer compatible with personal data. Therefore, it is imperative (even legally required) to design efficient redactable blockchain protocols in a controlled way.

In this paper, we present a generic approach of designing redactable blockchain protocol in the permissionless setting with instant redaction, applied to both proof-of-stake (PoS) blockchain and proof-of-work (PoW) blockchain with just different instantiations to randomly select "committees" according to stake or computational power. Our protocol can maintain the same adversary bound requirements and security assumption as the underlying blockchain (e.g., 1/2 adversary bound and asynchronous networks), which is compatible with most current blockchains requiring only minimal changes. It also offers public verifiability for redactable chains, where any edited block in the chain is publicly verifiable. Compared to previous solutions in permissionless setting, our redaction operation can be completed instantly, even only within one slot for the best-case scenario of PoS instantiation, which is desirable for redacting harmful or sensitive data. Correspondingly, our redaction verification in the blockchain is also instant. Furthermore, we define the first ideal functionality of redactable blockchain following the language of universal composition, and prove that our protocol can achieve the security property of redactable common prefix, chain quality, and chain growth. Finally, we develop a proof-of-concept implementation, and conduct extensive experiments to evaluate the overhead incurred by redactions. The experimental results show that the overhead remains minimal for both online nodes and re-spawning nodes, which demonstrates the high efficiency of our design.

KEYWORDS

Blockchain; Proof-of-Stake; Proof-of-Work; Redactable Blockchain

1 INTRODUCTION

Blockchain has been gaining increasing popularity and acceptance by a wider community, which enables Internet peers to jointly maintain a ledger. One commonly mentioned feature of blockchain is immutability (or untamperability) in mass media, and immutability of blockchain is paramount to certain applications to ensure keeping persistent records. However, in many other applications, such strict immutability may not be desirable or even hinder a wider adoption for blockchain technology.

First, since everyone in the Internet is able to write to permissionless blockchain, some malicious users may abuse the ability to post arbitrary transaction messages [40]. It could be the case that the data stored on the ledger might be sensitive, harmful or illegal. For instance, Bitcoin blockchain contains leaked private keys [41], materials that infringe on intellectual rights [28], and even child sexual abuse images [34]. It is clear that allowing those contents to be publicly available for everyone to access is unacceptable. They may affect the life of people forever, and block broader blockchain applications [16] in areas involving data such as government records [9, 23] and social media [3, 10].

On the other hand, as a full node, maintaining the whole ledger will also bear with the burden of maintaining those potentially illicit contents, thus the risk of being prosecuted for possessing and distributing illicit information increases. Concerning about above liability, honest nodes may opt-out as a full node, which in turn hurts the security of permissionless blockchain itself.

Indeed, with the adoption of the new European Union's General Data Protection Regulation (GDPR) [7] in May 2018, it is no longer compatible with current blockchains such as Bitcoin and Ethereum [6] to record personal data. In particular, GDPR imposes the "Right to be Forgotten" as a key Data Subject Right [29], i.e., the data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay. How to facilitate wider adoption of blockchain while complying with new regulations on personal data becomes a natural challenge.

Second, in certain systems, some flexibility is necessary to hedge with user mistakes or accidents to protect the system integrity. For example, in database, a rollback is the operation which returns the database to some previous state [30]. One other example is misdirected payment. According to statistics, around a quarter of people have accidentally paid the wrong person [1]. The Payments Council (part of UK finance) introduced a voluntary code of conduct

for banks and building societies to follow when it comes to these misdirected payments. If a user who made the mistake notifies his bank fast enough, and provides clear evidence, "his bank will contact the receiving bank on his behalf to request the money isn't spent, so long as the recipient doesn't dispute the claim" [1]. In the centralized banking system, there may still exist options to reverse incorrect transactions, while if similar mistakes happen in decentralized cryptocurrencies, thing would become much more complicated even if it is ever feasible.

We would like to stress that though blockchain offers a more reliable trust model as no single entity can fully control the system, however, it by no means insists on a strict immutability as an inherent property that is derived from consensus.

In fact, when the notorious DAO vulnerability was exploited, 3,641,694 Ethers (worth of about 79 million US dollars) were stolen due to the flaws of Ethereum and DAO contract [31], the financial losses have to be resolved by patching the vulnerability and "rollback" via a hard fork (majority of the miners are suggested by the Ethereum developers to adopt a newer client and create a fork of the chain from a state before the vulnerable contract got deployed). Hard forks also happened before, e.g., for Bitcoin when upgrading its protocol [4]. Of course, hard forks are not desirable as they may split the community and are very costly to implement.

Following above discussions, there exists a strong need to redact content of blockchain in exceptional circumstances, as long as the redaction proposal is clearly examined and satisfies full transparency and accountability (not determined by any single entity, and sufficient confidence can be gained that at least some honest users have approved the proposal).

1.1 Related Work

There exist several works that start exploring feasible methods for redacting blockchain.

A straightforward approach is to initiate a hard fork, which essentially requires all community members to vote by action (whether to follow the new fork). Doing this sometimes brings the risk of dividing the community, e.g., Bitcoin has a dozen forks, each of which now forms its own community. Moreover, such a procedure is extremely costly and slow, which normally takes multiple months to finalize [8], and if the redaction needs to touch an ancient block, growing a longer fork may take even much longer.

Ateniese et al. [12] proposed the notion of redactable blockchain in the permissioned setting. They use a chameleon hash function [15] to compute hash pointer, when redacting a block, a collision for the chameleon hash function can be computed by a trusted party (e.g., the certificate authority) with access to the chameleon trapdoor key. By this way, the block data can be modified while maintaining the chain consistency [11, 24]. Later, in order to support fine-grained and controlled redaction of blockchain, Derler et al. [20] introduced the novel concept of policy-based chameleon hash, where anyone who possesses enough privileges to satisfy the policy can then find arbitrary collisions for a given hash.

Their solutions focus on the permissioned setting, while in permissionless setting, there is no single trusted entity and users can join and leave the system at any time, thus their solutions will suffer from scalability issues when sharing the trapdoor key among miners and computing a collision for the chameleon hash function by a multi-party computation protocol. Moreover, public accountability of redaction cannot be provided in their solutions, and users are not clear to when and where a redaction having occurred.

Puddu et al. [39] also presented a redactable blockchain, called μ chain. In μ chain, the sender of a transaction can encrypt some different versions of the transaction, denoted by "mutations", the decryption keys are secretly shared among miners, and the unencrypted version of a transaction is regarded as the active transaction. When receiving a request for redacting a transaction, miners first check it according to redaction policy established by the sender of the transaction, then compute the appropriate decryption key by executing a multi-party computation protocol, and finally decrypt the appropriate version of the transaction as a new active transaction. Unfortunately, the malicious users who establish redaction policy can escape redaction, or even break the stability of transactions by the influence among transactions. Moreover, μ chain also faces scalability problem when reconstructing decryption keys by the multi-party computation protocol.

Recently, Deuber et al. [21] proposed the first redactable blockchain protocol in the permissionless setting, which does not rely on heavy cryptographic protocols or additional trust assumption. Once a redaction is proposed by a user, the protocol starts a consensus-based voting period, and only after obtaining enough votes for approving the redaction, the edition is performed on the blockchain. Each user can verify whether a redaction proposal is approved by checking the number of votes on the chain. Similarly, Thyagarajan et al. [42] proposed a generic protocol called Reparo on top of any blockchain to perform redactions, where the block structure remains unchanged by introducing external data structures to store block contents.

Their solutions are elegant, however, the new joined user has to check all the blocks within the voting period to verify a redaction on the blockchain. More importantly, the voting period is very long, for example, 1024 consecutive blocks are required in their Bitcoin instantiation, which takes about 7 days to confirm and publish a redaction block. Nevertheless, in practice, it is inefficient to redact sensitive data after such a long time, and it is also difficult to ask newly joined users in the system maintain these redactions. In addition, the threshold of votes in their solutions relies on chain quality of underlying blockchain, concretely, if the threshold of votes approaches $1/2$ (as in their bitcoin instantiation), the chain quality also approaches $1/2$. However, according to [37], the chain quality is close to $1 - \frac{\rho}{1-\rho}$, where ρ is the fraction of computational power the adversary controls, and thus redactable blockchains [21] [42] actually tolerate $\rho < 1/3$ adversary.

1.2 Our Contributions

In the permissionless setting, it seems unreasonable to have a trusted party holding certain trapdoor to modify the chain (like in the permissioned setting [12]). It follows that we have to choose a committee to jointly make the decision. Indeed, existing works [21, 42] pick one committee member per block. For this reason, the redaction will be at least linear to $T.t$, where T is the committee size, and

t is the block generation time of the underlying blockchain. However, in order to ensure honest majority, the committee size has to be substantially large.

In this work, we aim to achieve redactable blockchains in the permissionless setting such that the redaction could be *instant*, which means that the redaction time is at most $c \cdot t$ for a small constant c. Ideally $c = 1$, and thus the redaction could be as fast as the underlying chain!

More specifically, our technical contributions are threefold.

Generic construction of blockchain with instant redaction. We present a *generic* approach to design blockchain with *instant* redaction. Observe that existing work emulates the Bitcoin design, viewing block generation as a random walk that eventually converges to the longest chain, thus directly binding the committee selection to the consensus (treating each block as a random draw of a peer) requires a long convergence time (large number of blocks). But in certain blockchain design (such as Algorand [26]), one may use each block to randomly draw a large number of committee members, then let the committee members to run BFT to determine next block.

Inspired by this simple observation, we proceed in two steps. First, we deviate from the previous path and directly use the underlying component relying on stake or computing power to select committees randomly among all parties, ensuring a sufficient fraction of committee members are honest. In particular, the functionality of the committee election is refined by the general functions Cmt and VerifyCmt. However, it is challenging to make Cmt and VerifyCmt suitable for different instantiations such as PoS and PoW. In our approach, whether a party being elected as the committee member is based on his voting period instead of his current slot, i.e., the first slot sl of his voting period as the input parameter of functions, which makes committee selection more generic.

Then in the second phase, each committee member would vote by signing on the hash of the candidate edited block and diffuse the vote (i.e., the signature as well as the proof of committee members) to the network. To avoid the impact of network delays and collect enough votes, we set the maximum time of collecting votes (a.k.a. voting period) to be w slots, which is independent of block generation time. The leader of current slot (during voting period) adds votes collected and corresponding succinct proofs to his block.

On a high level, any party can propose a candidate edited block B_j^* for B_j in the chain, and only committee members in the voting period can promptly process the edit request once receiving B_j^*, including voting for B_j^* and broadcasting their votes and corresponding proofs; the slot leader during the voting period adds these votes and proofs to its block data collected and proposes a new block; if votes are approved by the redaction policy (e.g., voting bound in the voting period), B_j is replaced by B_j^*.

Note that our redaction method can achieve instant redaction, if the underlying blockchain progresses fast, then redaction will also be fast. Moreover, for the new joined user, it is also fast to verify a redaction in the blockchain. Furthermore, our redactable protocol can tolerate an adversary with less than 50% computational power (or stake), which is optimal in the blockchain protocol. This also means our approach will not reduce the adversary bound requirements of all blockchain protocols. Our protocol also offers accountability for redaction, where any edited block in the chain is publicly verifiable. In addition, multiple redactions per block can be performed throughout the execution of the protocol.

Simulation based security analysis of redactable blockchain. To characterize the security properties of redactable blockchains more precisely and analyze them rigorously, we define for the first time the *ideal functionality* of a redactable blockchain in the simulation based paradigm. Our proof first considers an idealized functionality $\mathcal{F}_{tree}$ that keeps track of all valid chains at any moment, and then shows that any attack that succeeds in real-world protocol can be turned into an attack in the idealized $\mathcal{F}_{tree}$ model. In the idealized functionality $\mathcal{F}_{tree}$, we use $\mathcal{F}_{tree}$.committee query to obtain the committee members, and $\mathcal{F}_{tree}$.redact query to redact the blockchain under certain conditions. In fact, separating these two queries in our idealized functionality ensures generality and *instant* redaction of redactable protocol. Moreover, $\mathcal{F}_{tree}$ models the ability of voting period changing with w.

As a sanity check, we show that the ideal functionality indeed implies the redactable common prefix property defined in [21], and the usual chain quality and chain growth properties [25]. Essentially, the redactable common prefix property ensures that any edited block which violates original common prefix should satisfy the redaction policy $\mathcal{RP}$. However, different from the redaction policy in [21] considering the consecutive ℓ blocks as the redaction period (which is not suitable for *instant* redaction), our $\mathcal{RP}$ requires votes are embedded in at most w slots, where ℓ is the committee size and w is the number of slot in the voting period.

Instantiations and performance evaluation. We demonstrate that our construction is generic by presenting concrete instantiations of the general functions Cmt and VerifyCmt on PoS and PoW (in principle, we may also instantiate via proof of space). Our instantiations can achieve the optimal 1/2 adversary bound and moreover support various network environments even asynchronous network, and thus provide compatibility with the underlying blockchain. In PoS instantiation, we similarly leverage hash function or verifiable random function (VRF) to sample sufficient number of committee members according to stake distribution. Different from 1/3 committee adversary tolerance in Algorand [26], we ensure the majority of commmitee members are honest by changing the constraint conditions on the expected committee size T.

While in PoW instantiations, more cares are needed. First, different from PoS, an adversary during the procedure of collecting votes still can continue to solve the computational puzzle, which leads to much computational power than honest users. Second, if an adversary in PoW withholds some blocks, and once these blocks are put to the chain, the adversary may have more advantage of computing the corresponding puzzles in advance. Finally, in asynchronous network, the actual number of committee members is impossible to be determined ahead of time, and thus it is hard to choose the voting bound. To resolve these *instant* redaction challenges in PoW blockchain, we propose a new approach to design redactable PoW blockchain in asynchronous network, and committee members are elected by finding solutions to a properly chosen *easy* puzzle (i.e.,

Table 1: Comparison of our redaction solution with existing works

	system-scale MPC	network compatibility[1]	adversary tolerance for PoW[2]	public verifiability	redaction time/slots[3]	verification time of redaction/slots[4]
Ateniese et al. [12]	required	yes	1/2	no	N/A	N/A
Puddu et al. [39]	required	yes	1/2	no	N/A	N/A
Derler et al. [20]	required	yes	1/2	no	N/A	N/A
Deuber et al. [21]	not required	yes	1/3	yes	513	513
Thyagarajan et al. [42]	not required	yes	1/3	yes	513	513
Ours	not required	yes	1/2	yes	PoS: 1 PoW: ≤ 20	PoS: 1 PoW: ≤ 20

[1] Network compatibility implies that the redaction solution does not impose any network assumption on the underlying blockchain.
[2] We only list the required adversary tolerance in PoW setting, while in PoS setting, all of adversary tolerance is 1/2 and thus omitted in the table.
[3] We evaluate the time one redaction can be completed in the best-case, where N/A is the abbreviation for the phrase "Not Applicable". In [21] and [42], the voting period is ℓ (instantiated to 1024 slots), and in the best-case more than one half of slots (i.e., 513 slots) are needed for the redaction. In our PoS construction, the redaction can be completed within just one slot if the underlying network is well enough. While in our PoW construction, the selection of committee members is completed in r slots, where r is instantiated to 20 in Section V, thus in the best-case 20 slots are enough for one redaction. Note that for all solutions in the table, one completed redaction can only be stable on the chain after several new blocks have been generated, e.g. six blocks in Bitcoin.
[4] We evaluate the time one redaction can be verified in the best-case, and the analysis is similar to the above item.

bigger difficulty parameter D), so that during regular mining procedure many easier puzzle solutions will be produced as a byproduct. In particular, according to "no long block withholding" lemma [37, Lemma 6.10], we increase the rounds of committee election to guarantee honest majority of committee members, even though the adversary has extra time advantage to find easier puzzle solutions. We also set the expected committee size satisfying two conditions: 1) a sufficient fraction of committee members are honest; and 2) malicious committee members cannot generate enough votes and complete the redaction of blockchain.

In addition, we give detailed analysis of each instantiation, and all of them satisfy the condition that committee members are chosen randomly and honest fraction of committees are guaranteed. The comparison of our construction with some related redactable blockchains is also shown in Table 1.

We also develop a proof-of-concept (PoC) implementation of our redaction approach, and conduct extensive experiments to evaluate the overhead after applying our redaction mechanism. The results demonstrate the high efficiency of our design. In particular, compared to the underlying blockchain (which simulates Cardano SL), the overhead incurred by redactions remains minimal for both online nodes and re-spawning nodes. For the online nodes, they only have to face a cheap and constant overhead (i.e., an extra latency of 0.8 second) to validate a newcoming block including a proof on redaction and then perform corresponding editing. For the re-spawning nodes, they can efficiently validate a redactable chain despite of many edited blocks. For example, when less than 6.25% blocks are edited, the time of validating a redactable chain is nearly same to that of validating an immutable chain. Remarkably, even if in the extremely pessimistic case that half blocks are edited, the performance of validating such a redacted chain remains acceptable (about 5X more than validating an unedited chain).

2 FORMAL ABSTRACTION OF BLOCKCHAIN

In this section, we define the formal abstraction of a blockchain based on the approach of Garay et al. [25] and Pass et al. [37, 38].

2.1 Protocol Execution Model

We assume a protocol specifies a set of instructions for the interactive Turing Machines (also called parties) to interact with each other. The protocol execution is directed by an environment $\mathcal{Z}$, which activates a number of parties (either honest or corrupt). Honest parties faithfully follow the protocol's prescription, whereas corrupt parties are controlled by an adversary $\mathcal{A}$. We assume that honest parties can broadcast messages to each other. The adversary $\mathcal{A}$ cannot modify the content of messages broadcasted by honest parties, but it can *delay* or *reorder* messages arbitrarily as long as it eventually delivers all messages.

We follow the nice results on the foundation of blockchains [32, 33] to assume a global clock, which can be seen as an equivalent notion of the height of the latest chain (or more specifically, the latest slot number in the blockchain). Notation-wise, by $Time$, we denote that a blockchain node invokes the global clock to get the current time. A protocol's execution proceeds in atomic time units. At the beginning of every time unit, honest parties receive inputs from an environment $\mathcal{Z}$; while at the end of every time unit, honest parties send outputs to $\mathcal{Z}$. $\mathcal{Z}$ can spawn, corrupt, and kill parties during the execution as follows.

- The environment $\mathcal{Z}$ can *spawn* new parties that are either honest or corrupt any time during the protocol's execution.
- The environment $\mathcal{Z}$ can *corrupt* an honest party and get access to its local state.
- The environment $\mathcal{Z}$ can *kill* either an honest or a corrupt party i, and at this moment, i is removed from the protocol execution.

2.2 Blockchain Protocol

We recall basic definitions [19] of blockchain. There are n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and each party $\mathcal{P}_i$ possesses a public/secret key pair (pk_i, sk_i). Without loss of generality, we assume that the public keys $pk_1, \ldots, pk_n$ are known by all system users. The protocol execution is divided in time units, called slots. We denote a block to be of the form $B_j := (header_j, d_j)$, where $header_j = (sl_j, st_j, G(d_j), \pi_j)$ denotes the block header information, and d_j denotes the block data. In $header_j$, $sl_j \in \{sl_1, \cdots, sl_R\}$ is the slot number, st_j is the

hash of the previous block header denoted by $H(header_{j-1}), G(d_j)$[1] denotes the state of the block data, and π_j contains some special header data for the block (e.g., in PoS, it's a signature on $(sl_j, st_j, G(d_j))$ computed under the secret key of slot leader generating the block, while in PoW, it is a nonce for the puzzle of PoW). Here H and G denote two collision-resistent hash functions.

A valid blockchain *chain* relative to the genesis block B_0 is a sequence of blocks $B_1, \ldots, B_m$ associated with a strictly increasing sequence of slots, where B_0 contains auxiliary information and the list of parties identified by their respective public-keys. We use $\mathsf{Head}(chain)$ to denote the head of *chain* (i.e., B_m). In a basic blockchain protocol, the users always update their current chain to the longest valid chain they have seen so far. Let $\mathrm{eligible}(\mathcal{P}_i, sl)$ be a function that determines whether a party $\mathcal{P}_i$ is an eligible leader at the time slot sl, then $\mathcal{P}_i$ can create a block at sl and broadcast the updated *chain* if $\mathrm{eligible}(\mathcal{P}_i, sl) = 1$, where the leader election can be achieved according to specific blockchain protocol.

We use $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$ to denote a randomized execution of the blockchain protocol Π with security parameter λ, which contains the joint view of all parties (i.e., all their inputs, random coins and all messages sent and received) in the execution. We use $|\mathsf{view}|$ to denote the number of time units in the execution trace view, and $\mathsf{chain}_i^t(\mathsf{view})$ denote the output of party i to the environment $\mathcal{Z}$ at time unit t in view of extracted ideal blockchain chain. The notation $\mathsf{chain}[i]$ denotes i-th block of chain, $\mathsf{chain}[:l]$ denotes the prefix of chain consisting of the first l blocks, $\mathsf{chain}[l:]$ denotes all blocks at length l or greater, and $\mathsf{chain}[:-l]$ denotes the entire chain except for the trailing l blocks. Security properties of blockchain protocol are referred to Appendix B.

3 REDACTING BLOCKCHAIN

In this section we present a generic construction that converts a basic blockchain into redactable blockchain protocol. We also extend the redactable protocol to accommodate multiple redactions for each block in Appendix G.

3.1 Overview of Redactable Blockchain Protocol

We construct our redactable blockchain protocol Γ by modifying and extending the basic blockchain protocol. We assume that the fraction of the computational power (or stake) held by honest users in the blockchain is h (a constant greater than 1/2). We denote by w the needed slots for votes diffusion, where w can be selected based on specific environments to guarantee the votes can be received by all users after w slots with a greater probability. For every $sl \bmod w = 0$, we also use the slots between sl and $sl + w - 1$ to denote the voting period for the editing proposal. In addition, we assume that there is some application-specific function $\mathsf{Cmt}(chain, sl, \mathcal{P}, para)$ that examines whether $\mathcal{P}$ is the committee member in the voting period beginning from sl and outputs $(c, proof)$, where $para$ is an optional parameter in specific instantiations, c is the weight of $\mathcal{P}$ in the committee, $proof$ is committee member proof. Correspondingly, there is some application-specific function $\mathsf{VerifyCmt}(chain, sl, c, proof, para')$ to verify $(c, proof)$, where $para'$ is the public parameter related to specific applications. In the committee selected by Cmt, we set the fraction of the computational power (or stake) held by honest users is at least η ($\eta > 1/2$).

First, a redaction policy is introduced to determine whether an edit to the blockchain should be approved or not.

Definition 3.1. (Redaction Policy $\mathcal{RP}$). We say that an edited block B^* at the slot sl satisfies the redaction policy, i.e., $\mathcal{RP}(chain, B^*, sl) = 1$, if the number of votes on B^* during a voting period is more than a threshold value[2], where each block embedding votes is in $\mathsf{chain}[:-k_0]$, and k_0 is the common prefix parameter.

Next, in order to accommodate editable data, we extend the above block structure to be of the form $B := (header, d)$, where $header = (sl, st, G(d), ib, \pi)$ and the newly added item ib denotes the original state of the block data. Specifically, if a blockchain *chain* with $\mathsf{Head}(chain) = (header, d)$ is updated to a new longer blockchain $chain' = chain \| B'$, the newly created block $B' = (header', d')$ sets $header' = (sl', st', G(d'), ib', \pi')$ with $st' = H(header)$ and $ib' = G(d')$. Notice that in order to maintain the link relationships between an edited block and its neighbouring blocks, inspired by the work [21] we introduce ib to represent the initial and unedited state of block, i.e., $ib = G(d_0)$ if original block data is d_0 in the edited block $B = (header, d)$, where $header = (sl, st, G(d), ib, \pi)$.

Generally, a blockchain $chain = (B_1, \cdots, B_m)$ can be redacted by the following steps.

(1) **Proposing a redaction.** If a user wishes to propose an edit to block B_j in *chain*, he parses $B_j = (header_j, d_j)$ with $header_j = (sl_j, st_j, G(d_j), ib_j, \pi_j)$, replaces d_j with the new data d_j^*, and then broadcasts the candidate block $B_j^* = (header_j^*, d_j^*)$ to the network, where $header_j^* = (sl_j, st_j, G(d_j^*), ib_j, \pi_j)$, and d_j^* is the empty data if the user wants to remove all data from B_j.
(2) **Updating the editing pool.** Upon receiving B_j^* from the network, every party $\mathcal{P}_i$ first validates whether B_j^* is a valid candidate editing block, and stores it in his own editing pool $\mathcal{EP}$ if it is. Notice that each candidate editing block in the pool $\mathcal{EP}$ has a period of validity t_p. At the beginning of each new slot sl, every party $\mathcal{P}_i$ tries to update his own editing pool $\mathcal{EP}$. Specifically, for every candidate editing block B_j^* in $\mathcal{EP}$: (i) $\mathcal{P}_i$ checks whether B_j^* has expired or not, and if it is, $\mathcal{P}_i$ removes B_j^* from $\mathcal{EP}$; (ii) $\mathcal{P}_i$ computes $\mathcal{RP}(chain, B_j^*, sl_j)$, and if it outputs 1, $\mathcal{P}_i$ removes B_j^* from $\mathcal{EP}$.
(3) **Voting for candidate editing blocks.** For each candidate editing block B_j^* in $\mathcal{EP}$, $\mathcal{P}_i$ checks whether he has voting right in the current voting period, which is determined by $\mathsf{Cmt}(chain, \lfloor sl'/w \rfloor * w, \mathcal{P}_i, para)$, where sl' is the current slot, and $\lfloor sl'/w \rfloor * w$ denotes the first slot in the current voting period. If it outputs $(c, proof)$ and $c \neq 0$, $\mathcal{P}_i$ broadcasts $(c, proof)$ and the signature sig on $H(B_j^*)$ as his votes.
(4) **Proposing new blocks.** The slot leader of sl' creates a block and broadcasts *chain* in exactly the same manner as the basic blockchain, if his editing pool is empty. Otherwise, for the candidate block B_j^* in the editing pool, the leader tries to collect and validate the votes on B_j^* in the voting period by using

[1] In practice $G(d_j)$ means the Merkle root of the block data.

[2] The threshold value would be set according to different committee selection methods such that it is more than the maximum number of votes the adversary can produce.

sub-protocol collectVote (Figure 2). If collectVote returns vote-proof at slot sl', the leader of sl' adds vote-proof to his block data, creates a new block and broadcasts *chain*.

(5) **Editing a block.** For each candidate block B_j^* in the editing pool $\mathcal{EP}$, the users check whether $\mathcal{RP}(chain, B_j^*, sl_j) = 1$. If yes, they replace $chain[j]$ with B_j^* and remove B_j^* from $\mathcal{EP}$.

Redactable blockchain protocol offers public verifiability. Concretely, to validate a redactable chain, users first check each block exactly like in the underlying immutable blockchain protocol. Once a "broken" link between blocks is found, users check whether the link still holds for the old state information, and whether the redaction policy $\mathcal{RP}$ is satisfied. By this way, the redaction operation of blockchain can be verified. For example, in the blockchain *chain* = $(B_1,\cdots,B_m)$, if $st_j \neq H(header_{j-1})$ for $header_{j-1} = (sl_{j-1}, st_{j-1}, G(d_{j-1}), ib_{j-1}, \pi_{j-1})$, *chain* is valid only under the condition of $st_j = H(sl_{j-1}, st_{j-1}, ib_{j-1}, ib_{j-1}, \pi_{j-1})$ and $\mathcal{RP}(chain, B_{j-1}, sl_{j-1}) = 1$.

For presentation simplicity, we extend the structure of block headers in the underlying blockchains, but it is straightforward to perform engineering optimizations to maintain the same block structure between the old and the new nodes. The idea behind the "soft-fork" could be simple [42]: i) the upgraded blockchain node maintains two separate storages for the original blockchain and the modifications respectively, so the blockchain's upgrade does not have to change the structure of block headers at the end of new nodes; ii) all modification requests and approvals are sent to the blockchain by rephrasing existing script opcodes, for example, through being attached to OP_RETURN in bitcoin-like script (e.g., Cardano's settlement layer).

3.2 Redactable Blockchain Protocol

Before our protocol is described, we first define the format of valid blocks, valid blockchains, and valid candidate editing blocks. Roughly speaking, we need to ensure that for an edited block, its original state before editing still can be accessible for verification.

Valid Blocks. To validate a block B, the validateBlock algorithm (Algorithm 1) first checks the validity of data included in B according to the system rules. It then checks the validity of the leader by eligible function. Finally, it verifies the signature π (on $(sl, st, G(d), ib)$ or on (sl, st, ib, ib)) with the public key pk of the leader or verifies the nonce π for the puzzle of PoW. In particular, for an edited block, the signature π is on the "old" state (sl, st, ib, ib). We say that B is a valid block iff validateBlock(B) outputs 1.

Algorithm 1 Block validation algorithm validateBlock(B)

1: Parse $B = (header, d)$, where $header = (sl, st, G(d), ib, \pi)$;
2: Validate data d, **if** invalid **return** 0;
3: Validate the leader, **if** invalid **return** 0;
4: Validate data π, **if** invalid **return** 0;
5: **else return** 1;

Valid Blockchains. To validate a blockchain *chain*, the validateChain algorithm (Algorithm 2) first checks the validity of every block B_j, and then checks its relationship to the previous block B_{j-1}, which has two cases depending on whether B_{j-1} is an edited block. If B_{j-1} has been redacted (i.e., $st_j \neq H(header_{j-1})$, its check additionally depends on whether the redaction policy $\mathcal{RP}$ has been satisfied. We say *chain* is valid iff validateChain(*chain*) outputs 1.

Algorithm 2 Chain validation algorithm validateChain(*chain*)

1: Parse $chain = (B_1, \cdots, B_m)$, parse $B_j = (header_j, d_j)$ where $header_j = (sl_j, st_j, G(d_j), ib_j, \pi_j)$, and set $j = m$;
2: **while** $j \geq 2$ **do**
3: **if** validateBlock(B_j) = 0,**return** 0;
4: **else if** $st_j = H(header_{j-1})$, **then** $j = j - 1$;
5: **else if** $st_j = H(sl_{j-1}, st_{j-1}, ib_{j-1}, ib_{j-1}, \pi_{j-1}) \wedge$
6: $\mathcal{RP}(chain, B_{j-1}, sl_{j-1}) = 1$, **then** $j = j - 1$;
7: **else return** 0.
8: **end while**
9: **return** validateBlock(B_j).

Valid Candidate Editing Blocks. To validate a candidate editing block B_j^* for the j-th block of blockchain *chain*, the validateCand algorithm (Algorithm 3) first checks the validity of B_j^*. It then checks the link relationship with B_{j-1} and B_{j+1}, where the link with B_{j+1} is "old", i.e., $st_{j+1} = H(sl_j, st_j, ib_j, ib_j, \pi_j)$. We say that B_j^* is a valid candidate editing block iff validateCand($chain, B_j^*$) outputs 1.

Algorithm 3 Candidate block validation algorithm validateCand(C, B_j^*)

1: Parse $B_j^* = (header_j, d_j^*)$, where $header_j = (sl_j, st_j, G(d_j^*), ib_j, \pi_j)$;
2: **if** validateBlock(B_j^*) = 0 **then return** 0;
3: Parse $B_{j-1} = (header_{j-1}, d_{j-1})$,
4: where $header_{j-1} = (sl_{j-1}, st_{j-1}, G(d_{j-1}), ib_{j-1}, \pi_{j-1})$;
5: Parse $B_{j+1} = (header_{j+1}, d_{j+1})$,
6: where $header_{j+1} = (sl_{j+1}, st_{j+1}, G(d_{j+1}), ib_{j+1}, \pi_{j+1})$;
7: **if** $st_j = H(sl_{j-1}, st_{j-1}, ib_{j-1}, ib_{j-1}, \pi_{j-1})$
8: and $st_{j+1} = H(sl_j, st_j, ib_j, ib_j, \pi_j)$, **then return** 1;
9: **else return** 0.

We now present redactable blockchain protocol Γ in Figure 1, where collectVote is used to collect the votes.

Collecting votes. The subroutine collectVote (Figure 2) collects and validates the votes from the slot sl (where sl mod $w = 0$) to the slot $sl + w - 1$. The collected votes are stored in *msgs* buffer. The algorithm first checks whether the number of votes on $H(B_j^*)$ is enough by $\mathcal{RP}(chain, B_j^*, sl_j)$, and stops collecting if it is. Otherwise, it begins to validate the vote. Specifically, it first verifies the signature on $H(B_j^*)$ under the public key of the voter, and then confirms the voting right and the voting number c of the voter determined by VerifyCmt($chain, sl, c, proof, para'$)[3]. Then the algorithm generates an aggregate signature $asig_j$ on all these valid vote signatures *SIG*, aggregates corresponding proofs *PROOF*, and returns them, where aggregate signature can reduce the communication complexity and storage overhead for blockchains.

4 SECURITY ANALYSIS

In this section, we analyze the security of redactable blockchain protocol Γ as depicted in Figure 1. The security properties of redactable

[3]In this paper, we assume the identifier of the public key would be sent to receivers associated with the signature, such that the corresponding public key can be located for verification.

Redactable Blockchain Protocol Γ (of Node $\mathcal{P}$)

/ * Initialization * /

Upon receiving init() from $\mathcal{Z}$, $\mathcal{P}$ is activated to initialize as follows:

let $(pk_p, sk_p) := \mathsf{Gen}(1^\lambda)$

/ / For simpler presentation, VRF uses the same keys

let $txpool$ be an empty FIFO buffer

let $chain := B_0$, where B_0 is the genesis block

let $\mathcal{EP}$ be an empty set (to store editing candidates)

let $\mathcal{VEP}$ be an empty set (to store proof for voted editings)

let $vote\_msgs$ be an empty FIFO buffer (to store votes)

/ * Receiving a longer chain * /

Upon receiving $chain'$ for the first time, the (online) $\mathcal{P}$ proceeds as:

assert $|chain'| > |chain|$ and $\mathsf{validateChain}(chain') = 1$;

let $chain := chain'$ and **broadcast** $chain$

/ * Receiving transactions * /

Upon receiving transactions(d') from $\mathcal{Z}$ (or other nodes) for the first time, the (online) $\mathcal{P}$ proceeds as:

let $txpool.enqueue(d')$ and **broadcast** d'

/ * Receiving candidate blocks for editing * /

Upon receiving edit(B_j^*) from $\mathcal{Z}$ (or other nodes) for the first time, the (online) $\mathcal{P}$ proceeds as:

let $\mathcal{EP} := \mathcal{EP} \cup \{B_j^*\}$, **if** $\mathsf{validateCand}(chain, B_j^*) = 1$

/ * Receiving vote information * /

Upon receiving $\mathsf{vote}(c_i, proof_i, pk_i, H(B_j^*), sig_j)$ for the first time, the (online) $\mathcal{P}$ proceeds as:

let $vote\_msgs.enqueue((c_i, proof_i, pk_i, H(B_j^*), sig_j))$

/ * When collectVote subroutine returns * /

Upon receiving vote-proof(v) from collectVote($sl, \ldots$) through the subroutine tape, the (online) $\mathcal{P}$ proceeds as:

let $\mathcal{VEP} := \mathcal{VEP} \cup v$, where v is in form of $(H(B_j^*), asig_j, PROOF)$

/ * Main procedure * /

for each slot $sl' \in \{1, 2, \ldots\}$, the (online) $\mathcal{P}$ proceeds as:

- **for** each B_j^* in $\mathcal{EP}$:
  - **if** B_j^* is expired, **let** $\mathcal{EP} := \mathcal{EP} \setminus \{B_j^*\}$
  - **if** $\mathcal{RP}(chain, B_j^*, sl_j) = 1$, **let** $chain[j] := B_j^*$, $\mathcal{EP} := \mathcal{EP} \setminus \{B_j^*\}$
- **if** $\mathcal{EP} \neq \emptyset$:
  - **let** $sl := \lfloor sl'/w \rfloor * w$
  - **activate** collectVote(sl, $vote\_msgs$, ...) subroutine
  - **let** $(c, proof) := \mathsf{Cmt}(chain, sl, \mathcal{P}, para)$
  - **if** c is non-zero:
    - **for** each B_j^* in $\mathcal{EP}$, **broadcast** $\mathsf{vote}(c, proof, pk_\mathcal{P}, H(B_j^*), sig_j)$, where $sig_j = \mathsf{Sign}(sk_\mathcal{P}; H(B_j^*))$
- **if** $\mathsf{eligible}(\mathcal{P}, sl') = 1$:
  - **let** $d' := txpool.dequeue() \cup \mathcal{VEP}$
  - **let** $(header, d) := \mathsf{Head}(chain)$
  - **let** $header' := (sl', st', G(d'), ib', \pi')$, where $st' := H(header)$ and π' is the output of $\mathcal{P}$ (the signature or the nonce)
  - **let** $chain := chain \| (header', d')$
  - **let** $\mathcal{VEP} := \emptyset$
  - **broadcast** $chain$
- **output** $\mathsf{extract}(chain)$ to $\mathcal{Z}$, where extract outputs an ordered list of each block in $chain$

Figure 1. Redactable Blockchain Protocol Γ

blockchain are same as that of basic blockchain, except for the common prefix property (c.f. Appendix B).

subroutine collectVote($chain$, sl, $msgs$, w, T, η) invoked by $\mathcal{P}$

/ / $msgs$ is a FIFO buffer keeping on receiving votes from the network

/ / sl is the number of the first slot in this w-slot voting period

let SIG be a dictionary of hash-set pairs;

let $PROOF$ be a dictionary of hash-set pairs;

Upon $Time^1 \geq sl + w$:

- **halt**

Upon $msgs$ not empty:

- **assert** $sl \leq Time < sl + w$
- **for** each $Time$
  - **for** each $(c, proof, pk, H(B_j^*), sig_j) \leftarrow msgs.dequeue()$
    - **if** $\mathcal{RP}(chain, B_j^*, sl_j) = 1$ **continue**;
    - **if** $SIG[B_j^*]$ and $PROOF[B_j^*]$ not initialized yet
      - **let** $SIG[B_j^*] := \emptyset$, $PROOF[B_j^*] := \emptyset$;
    - **if** sig_j on $H(B_j^*)$ cannot be validated by pk **continue**;
    - **if** $\mathsf{VerifyCmt}(chain, sl, c, proof, para') = 0$ **continue;**
    - $SIG[H(B_j^*)] := SIG[H(B_j^*)] \cup \{sig_j\}$;
    - $PROOF[H(B_j^*)] := PROOF[H(B_j^*)] \cup \{proof\}$;
  - **compute** aggregate signature $asig_j$ on $H(B_j^*)$ from $SIG[H(B_j^*)]$
  - **send** vote-proof($H(B_j^*), asig_j, PROOF[H(B_j^*)]$) to $\mathcal{P}$
  - **let** $SIG[H(B_j^*)] := \emptyset$ and $PROOF[H(B_j^*)] := \emptyset$

[1] $Time$ represents to invoke the global clock to get the latest slot number

Figure 2. Collecting Votes

Redactable Common Prefix. We observe that our protocol Γ inherently does not satisfy the original definition of common prefix due to the (possible) edit operation. In detail, consider the case where the party $\mathcal{P}_1$ is honest at time slot sl_1 and the party $\mathcal{P}_2$ is honest at time slot sl_2 in view, such that $sl_1 < sl_2$. For a candidate block B_j^* to replace the original B_j, whose votes are published at slot sl such that $sl_1 < sl < sl_2$, the edit request has not been proposed in $\mathsf{chain}_{\mathcal{P}_1}^{sl_1}(\mathsf{view})$ but may have taken effect in $\mathsf{chain}_{\mathcal{P}_2}^{sl_2}(\mathsf{view})$. As a result, the original B_j remains unchanged in $\mathsf{chain}_{\mathcal{P}_1}^{sl_1}(\mathsf{view})$ while it is replaced with the candidate B_j^* in $\mathsf{chain}_{\mathcal{P}_2}^{sl_2}(\mathsf{view})$. Therefore, $\mathsf{prefix}^k(\mathsf{view}) \neq 1$, which violates Definition B.1 in Appendix B.

The main reason lies in the fact that the original definition of common prefix does not account for edits in the chain, while any edit may break the common prefix property. To address this issue, we introduce an extended definition called redactable common prefix and consider the effect of each edit operation, which is suitable for redactable blockchains. Roughly speaking, the property of redactable common prefix states that if the common prefix property is violated, it must be the case that there exist edited blocks satisfying the redaction policy $\mathcal{RP}$.

Let $\mathsf{redactprefix}^k(\mathsf{view}) = 1$ if for all time $t \leq t'$, and for all parties $\mathcal{P}_i, \mathcal{P}_{i'}$ such that $\mathcal{P}_i$ is honest at t and $\mathcal{P}_{i'}$ is honest at t' in view, one of the following conditions is satisfied:

(1) the prefixes of $\mathsf{chain}_{\mathcal{P}_i}^t(\mathsf{view})$ and $\mathsf{chain}_{\mathcal{P}_{i'}}^{t'}(\mathsf{view})$ consisting of the first $|\mathsf{chain}_{\mathcal{P}_i}^t(\mathsf{view})| - k$ records are identical, or

(2) for each B_j^* in the prefix of $\mathsf{chain}_{\mathcal{P}_{i'}}^{t'}(\mathsf{view})$ but not in the prefix of $\mathsf{chain}_{\mathcal{P}_i}^t(\mathsf{view})$ consisting of the first $|\mathsf{chain}_{\mathcal{P}_i}^t(\mathsf{view})| - k$ records, it must be the case that $\mathcal{RP}(chain, B_j^*, t_j) = 1$ where $t_j < t < t'$.

Definition 4.1. (Redactable Common Prefix). We say a blockchain protocol Π satisfies k_0-redactable common prefix, if for all $(\mathcal{A}, \mathcal{Z})$, there exists a negligible function negl such that for every sufficiently large $\lambda \in \mathbb{N}$ and every $k \geq k_0$ the following holds:

$$\Pr[\text{view} \leftarrow \text{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda) : \text{redactprefix}^{k}(\text{view}) = 1] \geq 1 - \text{negl}(\lambda).$$

Essentially, Γ behaves just like the underlying immutable blockchain protocol in Appendix C if there is no edit in the chain, and otherwise each edit must be approved by the redaction policy $\mathcal{RP}$. Therefore, we prove Γ preserves the same properties (or a variation of the property) of the underlying immutable blockchain protocol under the redaction policy $\mathcal{RP}$.

Theorem 4.2. **(Security of Γ).** *Assume that the signature scheme* SIG *is EUF-CMA secure, the aggregate signature scheme* ASIG *is unforgeable, the hash function H is collision-resistant, the function* Cmt *ensures the fraction of honest users (in terms of computational power or stake) in the committee is at least η, and the underlying immutable blockchain protocol in Appendix C satisfies k_0-common prefix, (k_0, μ)-chain quality, and τ-chain growth. Then, redactable blockchain protocol Γ satisfies the k_0-redactable common prefix, (k_0, μ)-chain quality, and τ-chain growth.*

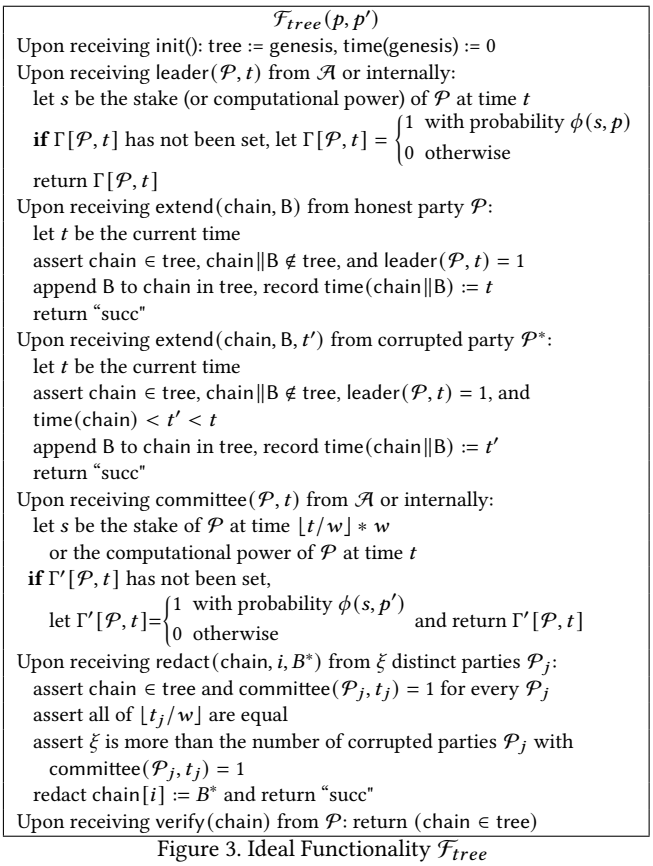
Proof roadmap. We first consider an ideal-world protocol Π_{ideal} having access to an ideal functionality $\mathcal{F}_{tree}$, and prove that Π_{ideal} satisfies redactable common prefix, chain quality, and chain growth in Section 4.1. Then we show that the real-world protocol Γ securely emulates the ideal-world protocol Π_{ideal} in Section 4.2.

4.1 Security of Ideal Protocol Π_{ideal}

We first define an ideal functionality $\mathcal{F}_{tree}$ (Figure 3) and analyze an ideal-world protocol Π_{ideal} (Figure 4) parameterized with $\mathcal{F}_{tree}$.

The ideal functionality $\mathcal{F}_{tree}$ keeps track of the set (denoted by tree) of all abstract blockchains mined so far. Initially, the only blockchain in the set tree is genesis. $\mathcal{F}_{tree}$ decides whether a party $\mathcal{P}$ is the elected leader for every time step t with probability $\phi(s, p)$ or the committee member with probability $\phi(s, p')$, where ϕ is a general function whose output is proportional to the stake (or the computational power) s of $\mathcal{P}$, and the parameter p (or p', resp.) provides the randomness. An adversary $\mathcal{A}$ can know which party is elected as the leader (or voting committee member, resp.) in time t using the $\mathcal{F}_{tree}$.leader (or $\mathcal{F}_{tree}$.committee, resp.) query. Further, honest and corrupted parties can extend known chains with new block by calling $\mathcal{F}_{tree}$.extend, if they are elected as leaders for specific time steps. Specifically, honest parties always extend chains in the current time, while corrupted parties are allowed to extend a malicious chain in a past time step t' as long as t' complies with the strictly increasing rule. In addition, the voting committee member can call $\mathcal{F}_{tree}$.redact to redact the blockchain, if the votes during one voting period are more than the number of corrupted committee members. Finally, $\mathcal{F}_{tree}$ keeps track of all valid chains, and parties can check if any chain they received is valid by calling $\mathcal{F}_{tree}$.verify.

Theorem 4.3. **(Security of Π_{ideal}).** *If the underlying immutable ideal protocol in Appendix D satisfies k_0-common prefix, (k_0, μ)-chain quality, and τ-chain growth, then Π_{ideal} satisfies the k_0-redactable common prefix, (k_0, μ)-chain quality, and τ-chain growth.*



$\mathcal{F}_{tree}(p, p')$

Upon receiving init(): tree := genesis, time(genesis) := 0

Upon receiving leader($\mathcal{P}, t$) from $\mathcal{A}$ or internally:
- let s be the stake (or computational power) of $\mathcal{P}$ at time t
- **if** $\Gamma[\mathcal{P}, t]$ has not been set, let $\Gamma[\mathcal{P}, t] = \begin{cases} 1 & \text{with probability } \phi(s, p) \\ 0 & \text{otherwise} \end{cases}$
- return $\Gamma[\mathcal{P}, t]$

Upon receiving extend(chain, B) from honest party $\mathcal{P}$:
- let t be the current time
- assert chain $\in$ tree, chain$\|$B $\notin$ tree, and leader($\mathcal{P}, t$) = 1
- append B to chain in tree, record time(chain$\|$B) := t
- return "succ"

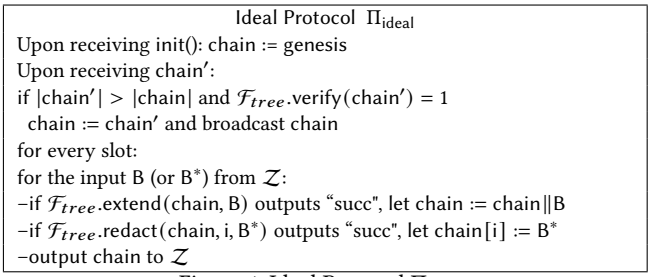
Upon receiving extend(chain, B, t') from corrupted party $\mathcal{P}^*$:
- let t be the current time
- assert chain $\in$ tree, chain$\|$B $\notin$ tree, leader($\mathcal{P}, t$) = 1, and time(chain) $< t' < t$
- append B to chain in tree, record time(chain$\|$B) := t'
- return "succ"

Upon receiving committee($\mathcal{P}, t$) from $\mathcal{A}$ or internally:
- let s be the stake of $\mathcal{P}$ at time $\lfloor t/w \rfloor * w$ or the computational power of $\mathcal{P}$ at time t
- **if** $\Gamma'[\mathcal{P}, t]$ has not been set, let $\Gamma'[\mathcal{P}, t] = \begin{cases} 1 & \text{with probability } \phi(s, p') \\ 0 & \text{otherwise} \end{cases}$ and return $\Gamma'[\mathcal{P}, t]$

Upon receiving redact(chain, i, B^*) from ξ distinct parties $\mathcal{P}_j$:
- assert chain $\in$ tree and committee($\mathcal{P}_j, t_j$) = 1 for every $\mathcal{P}_j$
- assert all of $\lfloor t_j/w \rfloor$ are equal
- assert ξ is more than the number of corrupted parties $\mathcal{P}_j$ with committee($\mathcal{P}_j, t_j$) = 1
- redact chain[i] := B^* and return "succ"

Upon receiving verify(chain) from $\mathcal{P}$: return (chain $\in$ tree)

Figure 3. Ideal Functionality $\mathcal{F}_{tree}$



Ideal Protocol Π_{ideal}

Upon receiving init(): chain := genesis

Upon receiving chain′:

if |chain′| > |chain| and $\mathcal{F}_{tree}$.verify(chain′) = 1
- chain := chain′ and broadcast chain

for every slot:

for the input B (or B*) from $\mathcal{Z}$:

–if $\mathcal{F}_{tree}$.extend(chain, B) outputs "succ", let chain := chain$\|$B

–if $\mathcal{F}_{tree}$.redact(chain, i, B*) outputs "succ", let chain[i] := B*

–output chain to $\mathcal{Z}$

Figure 4. Ideal Protocol Π_{ideal}

Proof Sketch. Note that if there is no edit in chain, then Π_{ideal} behaves exactly like the underlying immutable ideal protocol in Appendix D, and thus k_0-common prefix, (k_0, μ)-chain quality, and τ-chain growth can be preserved directly. Thus we mainly prove the security of Π_{ideal} with any edit satisfying the redaction policy $\mathcal{RP}$. We defer the security proof in Appendix E.

4.2 Real-world Emulates Ideal-world

We next show that the real-world protocol Γ as depicted in Figure 1 emulates the ideal-world protocol Π_{ideal}.

Theorem 4.4. **(Γ emulates Π_{ideal}).** *For any probabilistic polynomial-time (p.p.t.) adversary $\mathcal{A}$ of the real-word protocol Γ, there exists a p.p.t. simulator $\mathcal{S}$ of the ideal protocol Π_{ideal}, such that for any p.p.t.*

environment $\mathcal{Z}$, for any $\lambda \in \mathbb{N}$, we have:

$$\mathsf{view}(EXEC^{\Pi_{ideal}}(\mathcal{S}, \mathcal{Z}, \lambda)) \stackrel{c}{\equiv} \mathsf{view}(EXEC^{\Gamma}(\mathcal{A}, \mathcal{Z}, \lambda)),$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability.

Proof Sketch. The proof process can be shown by a standard simulation argument. Specifically, for any adversary $\mathcal{A}$ in the real world, we can construct a simulator $\mathcal{S}$ in the ideal world such that no p.p.t. environment $\mathcal{Z}$ can distinguish an ideal execution with the simulator $\mathcal{S}$ and Π_{ideal} from a real execution with the adversary $\mathcal{A}$ and Γ under the security assumption of the underlying primitives including the digital signature scheme, aggregate signature scheme and verifiable random function. We defer the (security) definitions of the corresponding primitives and security proof of the theorem in Appendix A and Appendix F respectively.

5 INSTANTIATION

Following the generic construction, we now present two concrete instantiations of redactable PoS blockchain and PoW blockchain.

5.1 Redactable Proof-of-Stake Blockchain

In proof-of-stake blockchain, we assume S is total stakes in the system, T is the expected number of stakes in committee for voting, and the fraction of stakes held by honest users in the committee is at least η. The committe members are selected only at the first slot sl of each voting period between sl and $sl + w - 1$, and w can be set based on specific network environment to guarantee the votes received by all users after w slots with a greater probability.

Checking committee members Cmt**.** The function Cmt (Algorithm 4) checks whether a party $\mathcal{P}_i$ (with secret key sk_i and stake s_i) is the committee member at the slot sl and outputs $(c, proof)$. Inspired by the idea of Algorand [26], Cmt uses VRFs to randomly select voters in a private and non-interactive way[4]. Specifically, $\mathcal{P}_i$ computes $(hash, \pi) \leftarrow VRF_{sk_i}(seed\|sl)$ with his own secret key sk_i, where $sl \bmod w = 0$, $seed$ is identical to that in the underlying proof-of-stake blockchain, and the pseudo-random $hash$ determines how many votes of $\mathcal{P}_i$ are selected. In order to select voters in proportion to their stakes, we regard each unit of stakes as a different "sub-user". For example, $\mathcal{P}_i$ with stakes s_i owns s_i units, each unit is selected with probability $p = \frac{T}{S}$, and the probability that q out of the s_i sub-users are selected follows the binomial distribution $B(q; s_i, p) = C(s_i, q)p^q(1-p)^{s_i-q}$, where $C(s_i, q) = \frac{s_i!}{q!(s_i-q)!}$ and $\Sigma_{q=0}^{s_i} B(q; s_i, p) = 1$. To determine how many sub-users of s_i in $\mathcal{P}_i$ are selected, the algorithm divides the interval [0,1) into consecutive intervals of the form $I^c = [\Sigma_{q=0}^{c} B(q; s_i, p), \Sigma_{q=0}^{c+1} B(q; s_i, p))$ for $c \in \{0, 1, \cdots, s_{i-1}\}$. If $\frac{hash}{2^{hashlen}}$ falls in the interval I^c, it means that c sub-users (i.e., c votes) of $\mathcal{P}_i$ are selected, where $hashlen$ is the bit-length of $hash$.

Verifying committee members VerifyCmt**.** The function VerifyCmt (Algorithm 5) verifies $\mathcal{P}_i$ (with public key pk_i) is the committee member with the weight c using $proof$ (i.e., $(hash, \pi)$). Specifically, it first verifies $proof$ by $\mathsf{VerifyVRF}_{pk_i}(hash, \pi, seed\|sl)$, and then verifies $\frac{hash}{2^{hashlen}}$ falls in the interval I^c.

[4]In a similar way, hash function can also be used to select committee members in a public way [18], which is secure against static adversary.

Algorithm 4 Checking committee members Cmt$(chain, sl, sk_i, s_i, seed, \mathcal{P}_i, T, S)$

1: $(hash, \pi) := VRF_{sk_i}(seed\|sl)$;
2: $p := \frac{T}{S}$; $c := 0$;
3: **while** $\frac{hash}{2^{hashlen}} \notin [\Sigma_{q=0}^{c} B(q; s_i, p), \Sigma_{q=0}^{c+1} B(q; s_i, p))$ **do**
4: $\quad c := c + 1$.
5: **end while**
6: $proof := (hash, \pi)$;
7: **return** $(c, proof)$.

Algorithm 5 Verifying committee members VerifyCmt$(chain, pk_i, sl, s_i, seed, c, proof, T, S)$

1: $(hash, \pi) := proof$;
2: **if** $\mathsf{VerifyVRF}_{pk_i}(hash, \pi, seed\|sl) = 0$, **then return** 0;
3: $p := \frac{T}{S}$; $\chi := 0$;
4: **while** $\frac{hash}{2^{hashlen}} \notin [\Sigma_{q=0}^{\chi} B(q; s_i, p), \Sigma_{q=0}^{\chi+1} B(q; s_i, p))$ **do**
5: $\quad \chi := \chi + 1$.
6: **end while**
7: **if** $\chi = c$, **then return** 1;
8: **else return** 0.

Parameter Selection. As mentioned earlier, we consider each unit of stakes as a different "sub-user", for example, if user U_i with s_i stakes owns s_i units, then U_i is regarded as s_i different "sub-users". We assume the total stakes S in the system is arbitrarily large. When a redaction is proposed, a committee for voting will be selected from all sub-users. The expected number of committee, T, is fixed, and thus the probability ρ_s of a sub-user to be selected is $\frac{T}{S}$. Then the probability that exactly K sub-users are sampled is

$$\begin{aligned}\binom{S}{K}\rho_s^K(1-\rho_s)^{S-K} &= \frac{S!}{K!(S-K)!}(\frac{T}{S})^K(1-\frac{T}{S})^{(S-K)} \\ &= \frac{S\cdots(S-K+1)}{S^K}\frac{T^K}{K!}(1-\frac{T}{S})^{(S-K)}\end{aligned}$$

If K is fixed, we have

$$\lim_{S\to\infty} \frac{S\cdots(S-K+1)}{S^K} = 1$$

and

$$\lim_{S\to\infty} (1-\frac{T}{S})^{(S-K)} = \lim_{S\to\infty} \frac{(1-\frac{T}{S})^S}{(1-\frac{T}{S})^K} = \frac{e^{-T}}{1} = e^{-T}$$

Then the probability of sampling exactly K sub-user approaches:

$$\frac{T^K}{K!}e^{-T} \tag{1}$$

Denote by $\#good$ and $\#bad$ the number of honest and malicious committee members respectively. If we set the majority of commmitee members are honest (i.e., $\eta > 1/2$), the following conditions should be satisfied.

(1): $\#good \geq 1/2 \cdot T$. The condition is violated when the number of honest committee members is $< 1/2 \cdot T$. From (1), the probability that we have exactly K honest committee members is $\frac{(h\cdot T)^K}{K!}e^{-h\cdot T}$, where honest stakes ratio in the system is h ($h > 1/2$). Thus, the probability of violating the condition is given by the formula:

$$\sum_{K=0}^{1/2\cdot T-1} \frac{(hT)^K}{K!}e^{-hT}.$$

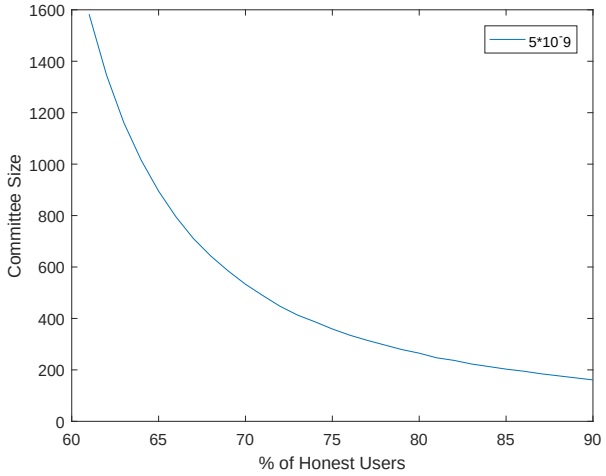


Figure 5. The x-axis specifies h, the stakes fraction of honest users. The y-axis specifies T, the committee size.

(2): $\#bad < 1/2 \cdot T$. As above, the probability that we have exactly L malicious committee members is $\frac{((1-h)\cdot T)^L}{L!} e^{-(1-h)\cdot T}$. Thus, the probability that satisfying the condition is given by the formula:

$$\sum_{L=0}^{1/2\cdot T-1} \frac{((1-h)T)^L}{L!} e^{-(1-h)T}.$$

F is a parameter which marks a negligible probability for failure of either condition, and our experience sets $F = 5{\times}10^{-9}$. Our goal is to minimize T, while maintaining the probability that conditions (1) or (2) fails to be at most F. If some value of T satisfies both conditions with probability $1 - F$, then any larger value of T also does with probability at least $1-F$. Based on the above observation, to find the optimal T, we firstly let T be an arbitrary large value, for example 10^4, and then check whether both conditions are satisfied. If both conditions are satisfied, we decrease T and check whether both conditions are still satisfied. We continue this process until finding the optimal T that ensures both conditions satisfied. In this way, we can get Figure 5, plotting the expected committee size T satisfying both conditions, as a function of h, with a probability of violation of 5×10^{-9}. A similar approach to compute the threshold of committee size can be referred to [26].

In the implementation of our system, we assume the fraction of honest stakes is 0.65, and thus we select $T = 1000$ according to Figure 5. A valid editing block is approved only when it obtains more than $1/2 \cdot T$ votes, that is, the threshold value in Definition 3.1 is equal to $1/2 \cdot T = 500$.

Fraction of Honest Users. According to Theorem 5.2, we only need to prove the fraction (in terms of stakes) of honest users in the committee is at least η. If $\mathcal{A}$ can "presciently" ensure which user would become the member of the voting committee, he can adaptively corrupt and impersonate this user, such that the fraction of honest users in the committee is less than η. However, according to the uniqueness property of the underlying VRF, the adversary has only a negligible probability $1/2^{hashlen}$ to win. In detail, the function value *hash* of VRF is random and unpredictable, the adversary without the secret key can only predict whether an honest user is chosen as the committee member with a negligible probability $1/2^{hashlen}$. In addition, $\mathcal{A}$ is allowed to corrupt the known committee members only after the corresponding w slots, which would not bring any non-negligible advantage since the committee would be reselected in the next voting period.

5.2 Redactable Proof-of-Work Blockchain

We also give an instantiation for PoW in asynchronous network. To get sufficient numbers of committee according to computational power distribution and ensure honest majority in the committees, we just need to collect sufficient PoW puzzle solutions. This can be easily realized by creating a "virtual selection" procedure using PoW with a bigger difficulty parameter D.

However, the adversary may be able to find "virtual puzzle solutions" in advance by the withholding attack. Specifically, if the adversary is lucky to produce a longer chain before sl that is likely to be the longest valid chain of slot sl, it temporarily withholds the chain and starts to find "virtual puzzle solutions". Then at slot sl, the adversary releases its chain and solutions, thus he has more time to find solutions. To thwart this attack, we elect the committee in r consecutive slots such that the majority of committee is honest even under the withholding attack. Like in the PoS instantiation, we use the network related parameter w to ensure all users would receive the votes with large probability, where $w \geq r$.

Checking committee members Cmt**.** In the function Cmt (Algorithm 6), if $\mathcal{P}$ can find some "virtual puzzle solutions" for PoW with difficulty parameter D between sl and $sl + r - 1$, $\mathcal{P}$ is elected as the committee and the weight c of $\mathcal{P}$ in the committee is the number of puzzle solutions. The committee member proof $proof$ includes the corresponding puzzle solutions.

Algorithm 6 Checking committee members Cmt$(chain, sl, pk, D, \mathcal{P}, r)$

1: $c := 0$;
2: $proof := \emptyset$;
3: $Time := sl$;
4: **while** $Time \leq sl + r - 1$ **do**
5: Parse $chain = (B_1, \cdots, B_m)$;
6: Parse $B_{Time} = (Time, pk, st, G(d), ib, \pi, d)$;
7: **if** $\mathcal{P}$ finds $nonce$ such that $H(Time, pk, st, G(d), nonce) < D$,
8: **then** $c := c + 1$, $proof := proof \cup (Time, pk, st, G(d), nonce)$;
9: **end while**
10: **return** $(c, proof)$.

Verifying committee members VerifyCmt**.** The function VerifyCmt (Algorithm 7) verifies whether $\mathcal{P}$ with the public key pk is the committee member by computing hash with the puzzle solutions, which is similar to Algorithm 6.

Parameter Selection. We assume the adversary is able to find "virtual puzzle solutions" at most t slots earlier than honest nodes and we elect the committee in r slots. Suppose that $h = \frac{1}{2} + \epsilon$ fraction of nodes in the underlying blockchain are honest, where $\epsilon \in (0, \frac{1}{2})$. Let $\alpha = \frac{D}{2^\ell} hn$ and $\beta = \frac{D}{2^\ell}(1-h)n$ denote the expected number of "virtual puzzle solutions" found by honest nodes and corrupt nodes in each slot respectively, where ℓ is the output length of the hash function $H(\cdot)$ and n is the total number of nodes.

Algorithm 7 Verifying committee members VerifyCmt($chain, pk, sl, D, c, proof, r$)

Parse $chain = (B_1, \cdots, B_m)$, where $B_i = (header_i, d_i)$, $i \in [1..m]$;
if the number of set member in $proof$ is not c **then return** 0;
for every proof in $proof$ **do**
  if $Time \geq sl + r$ **or** $Time < sl$, **then return 0;**
  if $H(Time, pk, st, G(d), nonce) \geq D$ **or** $st \neq H(header_{Time-1})$,
    then return 0;
end for
return 0.

We denote the maximum number of "virtual puzzle solutions" found by the adversary from the slot $sl - t$ to $sl + r - 1$ by N_A, and the minimum number of "virtual puzzle solutions" found by honest nodes from the slot sl to $sl + r - 1$ by N_H, respectively. Due to the Chernoff bound [17], for any $\delta > 0$, except with a negligible probability $p_1 = \exp(-\frac{\delta \cdot \min\{\delta,1\} \cdot \beta(t+r)}{3})$, it holds that $N_A \leq (1 + \delta)\beta(t + r)$. Similarly, for any $\delta \in (0, 1)$, except with a negligible probability $p_2 = \exp(-\frac{\delta^2 \alpha r}{2})$, it holds that $N_H \geq (1-\delta)\alpha r$. If we set the majority of committee members are honest (i.e., $\eta > 1/2$), then we need to guarantee $N_H > N_A$ and thus the following condition should be satisfied:

$$(1 + \delta)\beta(t + r) < (1 - \delta)\alpha r.$$

Therefore, we have $r > \frac{t}{\frac{(1-\delta)h}{(1+\delta)(1-h)} - 1}$.

According to "no long block withholding" lemma [37, Lemma 6.10], we set t to be the longest number of slots that the adversary can withhold a block B. Consider the case that k_0 new blocks are mined in the longest valid chain when the adversary withholds some blocks, where k_0 is the common prefix parameter. According to the common prefix property, these withholding blocks will never appear in the chains of honest nodes. Therefore, t should be less than the minimum time the longest valid chain increases by at least k_0 blocks. According to the chain growth property [37, Theorem 4.1], $t \approx k_0/\alpha'$, where $\alpha' = \frac{D'}{2^\ell} hn$ and D' is the difficulty parameter for the underlying PoW blockchain such that at least one party can find a puzzle solution at each slot (i.e., $\frac{D'}{2^\ell} n = 1$).

For instance, let $k_0 = 6$ as in Bitcoin, $h = 0.65$ and $\delta = 0.1$, then we have $r > 1.93t$ and without loss of generality we set $r = 2t$. Then we can compute $t = 10$ and $r = 20$. Further, if we set $p_1 = exp(-13)$ and $p_2 = exp(-25)$, then $D = \frac{5000}{hr} D' \approx 385D'$. An editing block would be approved only when it obtains more than $(1 + \delta)\beta(t + r) = (1 - h)(1 + \delta)\frac{5000}{hr}(t + r) \approx 4443$ votes, which are distributed among $r = 20$ slots, that is, the threshold value in Definition 3.1 is equal to $(1 + \delta)\beta(t + r) \approx 4443$.

6 IMPLEMENTATION AND EVALUATION

To demonstrate the feasibility of our approach, we choose redactable proof-of-stake blockchain just as an example and develop a proof-of-concept (PoC) implementation that simulates Cardano Settlement Layer (Cardano SL) [5]. We conduct extensive experiments on it, and reveal this non-optimized PoC implementation is already efficient. In particular, we showcase, even if in some extremely pessimistic cases (having tremendous redactions), the overhead of our approach remains acceptable (relative to an immutable chain).

6.1 Setup

Execution environment. We write in standard C language (C11 version) to implement a proof-of-stake chain that simulates Cardano SL (i.e., generating a valid local Cardano replica without executing consensus). The chain supports a subset of Cardano SL's bitcoin-style scripts, thus allowing to record basic ledger operations such as transacting coins and so on. Furthermore, we build our redaction protocol in it, thus enabling each block to include a special redaction transaction to solicit votes on editing earlier blocks. All tests are measured on a low-profile personal laptop installed with Ubuntu 16.04 (64bits) system, and equipped with a 2.20GHz Intel Core i5-5200U CPU and 8GB main memory.

Cryptographic building blocks. Our PoC implementation adopts ECDSA over secp256k1 for all digital signatures in both editing votes and block proposals, which is a widely adopted approach by PoC tests in the blockchain community [43]. For VRF, we adopt a generic approach due to deterministic "ECDSA" in the random oracle model [36]. We import the VRF's concrete instantiation over secp256k1 in C language from [2].

Other parameters. We set $h = 0.65$, namely, the adversary might control up to 35% of stakes in the system, which corresponds to the committee with expected size $T = 1000$. Moreover, when implementing Ouroboros Praos [19] (for simulating Cardano SL), we only consider one epoch, thus omitting the dynamic change of stakes. We might fix the block size in experiments. For example, we can specify that each block contains up to 10 transactions, which is enough to capture the number of transactions in nowadays Cardano. In addition, we also assume that each redaction request of editing a block only aims to modify a single transaction.

6.2 Experiments and measurements

Then we conduct extensive experiments in the above PoC "sandbox" to tell the small overhead of our redaction protocol relative to an immutable chain through various performance metrics.

Votes and proofs on redaction. As shown in Table 2, we begin with some preliminary experiments to understand (i) the generating time, the validating time, and the size of each vote on redaction as well as (ii) the validating time and the size of each proof on approved redaction. In general, these votes and proofs incur little computational burden and are also small in size, which at least flatters the necessary conditions of efficient redactions.

Table 2: Preliminary tests of votes and proofs on redaction

Vote on redaction candidate	Time to generate vote	~ 9 ms
	Time to validate vote	~ 1 ms
	Size of each vote	~ 0.2 KB
Proof on approved redaction	Time to validate proof	~ 560 ms
	Size of each proof	~ 109 KB

Proposing/receiving new blocks with redaction proof. To evaluate how redactions would impact the performance of consensus, we consider two key metrics in the *online* nodes' critical path: (i) the latency of producing new blocks with redaction and (ii) the latency of appending new blocks with redaction to the local replica.

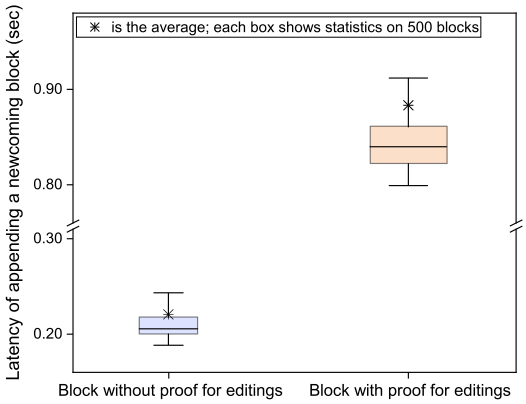


Figure 6. The latency of appending a newcoming block (without or with proof on redaction) to the local replica.

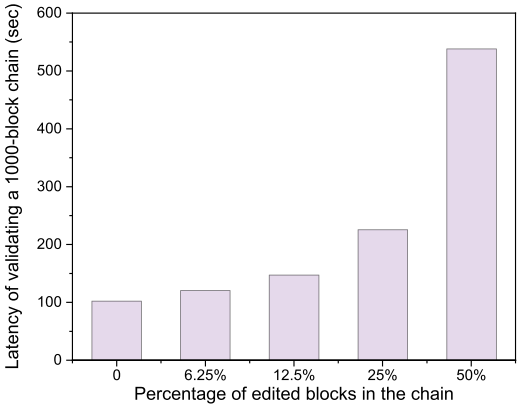


Figure 7. The latency of validating 1000-block redactable chains (respect to various percentages of editings).

First, we consider the latency of producing blocks with redaction proof(s) and without redaction proof(s), respectively. For both cases, we test 500 blocks (with fixed size up to 10 transactions), and do not realize any statistic differences. Nevertheless, this is not surprising, because we explicitly decouple the generation of blocks and the voting on redaction, so the generation of blocks in the two cases would execute the exactly same code.

Second, we measure the time spent on appending newly received blocks to the local storage, for the cases with redaction proof(s) and without redaction proof(s) respectively. As illustrated in Figure 6, we compare appending a block with a redaction proof to the benchmark case of appending a block without any redaction proof. For each case, we conduct extensive tests to get statistics on 500 blocks (at distinct slots but with fixed block size up to 10 transactions) and visualize the statistics. It reveals that the extra overhead (incurred by validating redaction proof and editing earlier block) is small and nearly constantly. In particular, compared to the immutable case, the node only needs an extra time of 0.7 second to (i) validate a redaction proof and (ii) edit an earlier block accordingly.

Validating a chain consisting of edited blocks. Then, we conduct a series of experiments to measure the extra cost of validating an entire chain with edited blocks. Comparing to validating the immutable chain, validating an edited chain further requires to fetch and validate the proof on redaction for each edited block (besides validating block headers). This could be another critical metric to reflect how efficient our scheme is regarding *re-spawning* nodes.

To this end, we evaluate the time needed to validate a redactable chain, with respect to the varying portion of edited blocks. In the experiments, we generate redactable chains consisting of 1000 blocks and each block contains 10 transactions, and measure the time to validate them. As shown in Figure 7, the latency of validating chains is almost increasing linearly in the number of redactions, especially when the percentage of edited blocks is small or moderately large (e.g., smaller than 25%). For example, when the percentage of edited blocks is 6.25% and 12.5%, the *extra* latency to verify the chain is about 10 seconds and 30 seconds, respectively. Even if in the extremely pessimistic case (i.e., 50% blocks are edited), the cost is still acceptable (i.e., about 5x the immutable case).

6.3 More discussions

Minimal impact on consensus. When proposing and receiving (new) blocks with proofs on redaction, there is only small overhead in our design. That means it places little burden on the *online* blockchain nodes, and more importantly, it causes minimal overhead to the critical path of consensus. In particular, when proposing new blocks with redaction, there is no extra cost to slow down the consensus; while receiving new blocks with redaction, the extra latency is as small as 0.8 second.

Efficiency for re-spawning nodes. When some nodes are re-spawning, they have to bootstrap to sync up to the current longest chain. Our extensive experiments reveal it would be feasible for the re-spawning node to download and then verify the entire chain despite of a few editable blocks. Especially, in the normal cases that edited blocks are rare (e.g., less than 6.25%), the extra cost incurred by redaction is overwhelmed by the original cost of validating chain headers and transactions.

Instant redaction (close to actual network delay). Our design dedicates to decouple voting from consensus: all votes are diffused across the network via the underlying gossip network; once the votes are successfully diffused, any honest block proposer can include a proof on redaction in its block, which would be confirmed immediately after the block becomes stable. This typically costs only a couple of minutes in Cardano. In contrast, prior art [21] lets the node proposing a block to embed its own vote in the block, resulting in a latency liner to a large security parameter. For example, [21] requires about 1024 consecutive blocks to collect votes, which means about 6 hours in Cardano and 7 days in Bitcoin. To sum up, our construction achieves significant improvement by greatly reducing the latency of confirming redactions.

Possible storage optimizations. Different from the immutable blockchain, our redaction protocol has to store the collected votes on each redaction, which is the most significant storage overhead relative to an immutable blockchain. Currently, our PoC implementation requires about 110 KB to store the votes for each redaction. We remark that various optimizations can be explored to further reduce the storage overhead. For example, we can use pairing-based

multi-signature scheme [13] to aggregate signatures of votes instead of trivially concatenating secp256k1 ECDSA, which can reduce the size of votes to only about 60 KB.

7 CONCLUSION

It is crucial and even legally required to design redactable blockchain protocols with instant redaction. We propose a generic approach to construct redactable blockchain protocols with instant redaction, where redactable blockchain inherits the same security assumption from the underlying blockchain. We also define the first ideal functionality of redactable blockchain following the language of universal composition, and prove the security of our construction. Moreover, we present concrete instantiations of redactable PoS and PoW blockchains. Finally, we develop a PoC implementation of our PoS instantiation, and the experimental results demonstrate the high efficiency of our design. Our work makes a step forward in understanding of redactable blockchain protocols.

REFERENCES

[1] https://www.lovemoney.com/news/91297/sent-money-to-the-wrong-account-get-money-back-after-misdirected-payment.
[2] https://github.com/aergoio/secp256k1-vrf.
[3] Akasha. https://akasha.world.
[4] All about the bitcoin cash hard fork. https://www.investopedia.com/news/all-about-bitcoin-cash-hard-fork.
[5] Cardano. https://cardano.org/.
[6] Ethereum project. https://www.ethereum.org/.
[7] The EU general data protection regulation. https://gdpr-info.eu/.
[8] The hard fork: what's about to happen to ethereum and the DAO. https://www.coindesk.com/hard-fork-ethereum-dao.
[9] The illinois blockchain initiative. https://illinoisblockchain.tech.
[10] Steem. https://steem.
[11] Giuseppe Ateniese, Michael T Chiaramonte, David Treat, Bernardo Magri, and Daniele Venturi. 2018. Rewritable blockchain. uS Patent 9,967,096.
[12] Giuseppe Ateniese, Bernardo Magri, Daniele Venturi, and Ewerton Andrade. 2017. Redactable blockchain - or - rewriting history in bitcoin and friends. In *IEEE European Symposium on Security and Privacy, EuroS&P 2017*. 111–126.
[13] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *ASIACRYPT 2018*, Vol. 11273. Springer, 435–464.
[14] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of Eurocrypt 2003*. Springer, 416–432.
[15] Jan Camenisch, David Derler, Stephan Krenn, Henrich C.Pohls, Kai Samelin, and Daniel Slamanig. 2017. Chameleon-hashes with ephemeral trapdoors. In *IACR International Workshop on Public Key Cryptography*. Springer, 152–182.
[16] CBinsights. 2018. Banking is only the beginning: 50 big industries blockchain could transform. https://www.cbinsights.com/research/ industries-disrupted-blockchain/.
[17] Herman Chernoff. 1952. A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics* 23 (1952), 493–509.
[18] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow White: robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security 2019*. Springer, 23–41.
[19] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Proceedings of Eurocrypt 2018*. Springer, 66–98.
[20] David Derler, Kai Samelin, Daniel Slamanig, and Christoph Striecks. 2019. Fine-grained and controlled rewriting in blockchains: chameleon-hashing gone attribute-based. In *Network and Distributed Systems Security (NDSS) Symposium 2019*.
[21] Dominic Deuber, Bernardo Magriy, Sri Aravinda, and Thyagarajan Krishnan. 2019. Redactable blockchain in the permissionless setting. In *IEEE Symposium on Security and Privacy 2019*. 124–138.
[22] Yevgeniy Dodis and Aleksandr Yampolskiy. 2005. A verifiable random function with short proofs and keys. In *8th International Workshop on Theory and Practice in Public Key Cryptography*. 416–431.
[23] The Economist. 2017. Governments may be big backers of the blockchain. https://goo.gl/uEjckp.
[24] Accenture files patent for editable blockchain. 2016. *Business Insider Deutschland*. https://tinyurl.com/yblq9zdp.
[25] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. 9057 (2015), 281–310.
[26] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 51–68.
[27] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* 17 (1988), 281–308.
[28] Steve Hargreaves and Stacy Cowley. 2013. How porn links and ben bernanke snuck into bitcoin's code. http://money.cnn.com/2013/05/02/technology/security/bitcoin-porn/index.html
[29] O'Hara Kieron Ibanez, Luis-Daniel and Elena Simperl. 2018. On blockchains and the general data protection regulation. In *Network and Distributed Systems Security (NDSS) Symposium 2019*.
[30] Michael Isard and MartÍÎn Abadi. Falkirk wheel: rollback recovery for dataflow systems. https://arxiv.org/abs/1503.08877.
[31] Christoph Jentzsch. Decentralized autonomous organization to automate governance. https://download.slock.it/public/DAO/WhitePaper.pdf.
[32] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. 2016. Fair and robust multi-party computation using a global transaction ledger. In *Eurocrypt (2) 2016*. Springer, 705–734.
[33] Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Symposium on Security and Privacy 2016*. 839–858.
[34] Jerin Mathew. 2015. Bitcoin: Blockchain could become 'safe haven' for hosting child sexual abuse images. http://www.dailydot.com/business/bitcoinchild-porn-transaction-code/.
[35] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 120–130.
[36] Dimitrios Papadopoulos, Duane Wessels, Shumon Huque, Moni Naor, Jan Včelák, Leonid Reyzin, and Sharon Goldberg. 2017. Making NSEC5 Practical for DNSSEC. Cryptology ePrint Archive, Report 2017/099. (2017). https://eprint.iacr.org/2017/099.
[37] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt 2017*, Vol. 10211. Springer, 643–673.
[38] Rafael Pass and Elaine Shi. 2017. The sleepy model of consensus. In *ASIACRYPT 2017*, Vol. 10625. Springer, 380–409.
[39] Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. 2017. μ chain: how to forget without hard forks. In *IACR Cryptology ePrint Archive, 2017/106*.
[40] Matzutt R, Hiller J, and Henze M. 2018. A quantitative analysis of the impact of arbitrary blockchain content on bitcoin. In *Financial Cryptography and Data Security 2018*. Springer, 420–438.
[41] Ken Shirriff. 2014. Hidden surprises in the bitcoin blockchain and how they are stored: Nelson mandela, wikileaks, photos, and python software. http://www.righto.com/2014/02/ascii-bernanke-wikileaks-photog raphs.html.
[42] S A Krishnan Thyagarajan, Adithya Bhat, Bernardo Magriz, Daniel Tschudix, and Kate Aniket. Reparo: publicly verifiable layer to repair blockchains. https://arxiv.org/abs/2001.00486.
[43] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2018. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069* (2018).

A PRELIMINARIES AND DEFINITIONS

In this paper, we say a function $negl(\cdot) : \mathbb{N} \rightarrow (0, 1)$ is negligible, if for every constant $c \in \mathbb{N}$, $negl(n) < n^{-c}$ for sufficiently large n. Hereafter, we use $negl(\lambda)$ to refer to a negligible function in the security parameter λ.

A.1 Signature Scheme

A digital signature scheme SIG = (Gen, Sign, Verify) with message space $\mathcal{M}(\lambda)$ consists of the standard algorithms: key generation $\text{Gen}\left(1^\lambda\right) \xrightarrow{\$} (pk, sk)$, signing $\text{Sign}(sk; m) \rightarrow \sigma$, and verification $\text{Verify}(pk; m, \sigma) \rightarrow \{0, 1\}$. It is said to be correct if $\text{Verify}(pk; m, \text{Sign}(sk; m)) = 1$ for all $(pk, sk) \xleftarrow{\$} \text{Gen}\left(1^\lambda\right)$ and $m \in \mathcal{M}(\lambda)$.

To define security [27], we consider the following game between an adversary $\mathcal{A}$ and a challenger.

(1) Setup Phase. The challenger chooses $(pk, sk) \xleftarrow{\$} \mathsf{Gen}\left(1^{\lambda}\right)$.
(2) Signing Phase. The adversary $\mathcal{A}$ sends signature query $m_i \in \mathcal{M}$ and receives $\sigma_i = \mathsf{Sign}(sk; m_i)$.
(3) Forgery Phase. $\mathcal{A}$ outputs a message m and its signature σ. If m is not queried during the Signing Phase and $\mathsf{Verify}(pk;\ m,\ \sigma) = 1$, the adversary wins.

Definition A.1. (EUF-CMA). We say that a signature scheme SIG is *existentially unforgeable under adaptive chosen-message attacks* (EUF-CMA), if for all adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathrm{Adv}_{\mathsf{SIG}}^{\mathsf{EUF\text{-}CMA}} = \Pr[\mathcal{A}\ wins] \leq \mathsf{negl}(\lambda).$$

A.2 Aggregate Signature Scheme

An aggregate signature scheme [14] allows aggregating multiple individual signatures into a single short signature in a non-interactive way.

An aggregate signature scheme ASIG consists of five algorithms: KeyGen, Sign, Ver, Agg and AggVer. The key generation algorithm $\mathsf{KeyGen}(1^{\lambda}) \xrightarrow{\$} (pk_i, sk_i)$ generates the public/secret key pair for each participant. The signing algorithm $\mathsf{Sign}(sk, m) \rightarrow \sigma$ generates a signature σ on the message m using the secret key sk. The verification algorithm $\mathsf{Ver}(pk, m, \sigma)$ outputs 1 if σ is a valid signature on m under pk, otherwise outputs 0. Given multiple individual signatures $(\sigma_1, ..., \sigma_n)$, where σ_i is a signature on the message m_i under pk_i for $i \in [n]$, the aggregation algorithm $\mathsf{Agg}((pk_1, m_1, \sigma_1),...,(pk_n, m_n, \sigma_n)) \rightarrow asig$ aggregates these signatures into one signature $asig$. The aggregate verification algorithm $\mathsf{AggVer}(\{(pk_1, m_1), ... ,(pk_n,m_n)\}, asig)$ outputs 1 if $asig$ is a valid aggregate signature on $(m_1, ..., m_n)$ under $(pk_1, ..., pk_n)$, otherwise outputs 0.

An aggregate signature scheme should satisfy completeness, which means that for any n, $\{(pk_1, sk_1), ..., (pk_n, sk_n)\} \leftarrow \mathsf{KeyGen}(1^{\lambda})$, any distinct messages $\{m_1, , ..., m_n\}$, $\sigma_i \leftarrow \mathsf{Sign}(sk_i, m_i)$ for $i \in [n]$, and $asig \leftarrow \mathsf{Agg}((pk_1, m_1, \sigma_1), ..., (pk_n, m_n, \sigma_n))$, we have $\mathsf{AggVer}(\{(pk_1, m_1),...,(pk_n, m_n)\}, asig) = 1$ if $\mathsf{Ver}(pk_i, m_i, \sigma_i) = 1$ for $i \in [n]$.

An aggregate signature scheme should also satisfy unforgeability. To define unforgeability, we consider the following game between an adversary $\mathcal{A}$ and a challenger.

(1) Setup Phase. The challenger generates the challenge public/secret key pair $(pk^*, sk^*) \leftarrow \mathsf{KeyGen}(1^{\lambda})$, and sends pk^* to $\mathcal{A}$.
(2) Signing Phase. $\mathcal{A}$ can make signature queries on any message m under pk^*, and the challenger returns $\sigma \leftarrow \mathsf{Sign}(sk^*, m)$.
(3) Forgery Phase. $\mathcal{A}$ outputs a public key set $PK = \{pk_1, ..., pk_{n-1}\}$, a message set $M = \{m^*, m_1, ..., m_{n-1}\}$ and an aggregate signature $asig$. If $pk^* \in PK$, m^* is not queried to $\mathsf{Sign}(sk^*, .)$, and $\mathsf{AggVer}(\{(pk^*, m^*), (pk_1, m_1), ..., (pk_{n-1}, m_{n-1})\}, asig) = 1$, the adversary wins.

Definition A.2. (Unforgeability). We say that an aggregate signature scheme ASIG is *unforgeable*, if for all adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\mathrm{Adv}_{\mathsf{ASIG}} = \Pr[\mathcal{A}\ wins] \leq \mathsf{negl}(\lambda).$$

A.3 Verifiable Random Functions

The concept of verifiable random functions is introduced by Micali et al.[35]. Informally, it is a pseudo-random function that provides publicly verifiable proofs on outputs correctness.

Definition A.3. (Verifiable Random Functions)[22]. A function family $F_{(\cdot)}(\cdot) : \{0,1\}^{l} \rightarrow \{0,1\}^{l_{VRF}}$ is a family of VRFs if there exist algorithms (Gen, VRF, VerifyVRF) such that Gen outputs a pair of keys (pk, sk); $\mathsf{VRF}_{sk}(x)$ outputs a pair $(F_{sk}(x), \pi_{sk}(x))$, where $F_{sk}(x)$ is the output value of the function and $\pi_{sk}(x)$ is the proof for verifying correctness; and $\mathsf{VerifyVRF}_{pk}(x, y, \pi)$ verifies that $y = F_{sk}(x)$ using the proof π, return 1 if y is valid and 0 otherwise. Formally, we require the following properties:

- Uniqueness: no values $(pk,\ x,\ y_1,\ y_2,\ \pi_1,\ \pi_2)$ can satisfy $\mathsf{VerifyVRF}_{pk}(x, y_1, \pi_1) = \mathsf{VerifyVRF}_{pk}(x, y_2, \pi_2)$ unless $y_1 = y_2$.
- Provability: if $(y, \pi) = \mathsf{VRF}_{sk}(x)$, then $\mathsf{VerifyVRF}_{pk}(x, y, \pi) = 1$.
- Pseudorandomness: for any probabilistic polynomial time algorithm $A = (A_E, A_J)$, which executes for a total of $s(\lambda)$ steps when its first input is 1^{λ}, and does not query the oracle on x,

$$\Pr\left[b = b' \middle| \begin{array}{l} (pk, sk) \leftarrow \mathsf{Gen}(1^{\lambda}); \\ (x, st) \leftarrow A_E^{VRF(.)}(pk); \\ y_0 = \mathsf{VRF}_{sk}(x); y_1 \leftarrow \{0,1\}^{l_{\mathrm{VRF}}}; \\ b \leftarrow \{0,1\}; b' \leftarrow A_J^{VRF(.)}(y_b, st) \end{array}\right] \leq \frac{1}{2} + negl(\lambda).$$

Intuitively, the pseudorandomness property states that no function value can be distinguished from random, even after seeing any other function values together with corresponding proofs.

B SECURITY PROPERTIES OF BLOCKCHAIN

We recall common security properties that blockchain protocols should satisfy as follows.

Common Prefix. Informally speaking, the common prefix property requires that all honest parties' chains should be identical except for roughly $O(\lambda)$ number of trailing blocks that have not stabilized.

Let $\mathsf{prefix}^k(\mathsf{view}) = 1$ iff for all times $t \leq t'$, and for all parties i, j such that i is honest at t and j is honest at t' in view, we have that the prefixes of $\mathsf{chain}_i^t(\mathsf{view})$ and $\mathsf{chain}_j^{t'}(\mathsf{view})$ consisting of the first $|\mathsf{chain}_i^t(\mathsf{view})| - k$ records are identical.

Definition B.1. (Common Prefix). We say that a blockchain protocol Π satisfies k_0-common prefix, if for all $(\mathcal{A}, \mathcal{Z})$, there exists a negligible function negl such that for every sufficiently large $\lambda \in \mathbb{N}$ and every $k \geq k_0$ the following holds:

$$\Pr[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathsf{prefix}^k(\mathsf{view}) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

Chain Quality. Informally speaking, the chain quality property requires that the ratio of adversarial blocks in any segment of a chain held by an honest party is not too large.

We say that a block $B = \mathsf{chain}[j]$ is honest w.r.t. view and prefix $\mathsf{chain}[:\ j']$ where $j' < j$, if there exists some honest party i at some time $t < |\mathsf{view}|$ who received B as input, and its local chain $\mathsf{chain}_i^t(\mathsf{view})$ contains the prefix $\mathsf{chain}[:\ j']$.

Let $\mathsf{quality}^k(\mathsf{view}, \mu) = 1$ iff for every time t and every party i such that i is honest at t in view, among any consecutive sequence

of k blocks $\mathsf{chain}[j{+}1..j{+}k] \subseteq \mathsf{chain}_i^t(\mathsf{view})$, the fraction of blocks that are honest w.r.t. view and prefix $chain[:j]$ is at least μ.

Definition B.2. (Chain Quality). We say that a blockchain protocol Π satisfies (k_0, μ)-chain quality, if for all $(\mathcal{A}, \mathcal{Z})$, there exists a negligible function negl such that for every sufficiently large $\lambda \in \mathbb{N}$ and every $k \geq k_0$ the following holds:

$$\Pr[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathsf{quality}^k(\mathsf{view}, \mu) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

Chain Growth. The chain growth property requires that the chain grows proportionally with the number of time slots. Let $\mathsf{growth}^{\tau}(\mathsf{view}) = 1$ iff for every time $t \leq |\mathsf{view}| - t_0$ and every two parties i, j such that in view i is honest at time t and j is honest at $t + t_0$, $|\mathsf{chain}_j^{t+t_0}(\mathsf{view})| - |\mathsf{chain}_i^t(\mathsf{view})| \geq \tau \cdot t_0$.

Definition B.3. (Chain Growth). We say that a blockchain protocol Π satisfies τ-chain growth, if for all $(\mathcal{A}, \mathcal{Z})$, there exists a negligible function negl such that for every sufficiently large $\lambda \in \mathbb{N}$ the following holds:

$$\Pr[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathsf{growth}^{\tau}(\mathsf{view}) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

C IMMUTABLE BLOCKCHAIN PROTOCOL

We now recall the immutable blockchain protocol Γ' in Figure 8. Compared with the redactable protocol Γ as depicted in Figure 1, the redaction operations are pruned and the original block structure is adopted.

Immutable Blockchain Protocol Γ' (of Node $\mathcal{P}$)

/ * Initialization * /
Upon receiving init() from $\mathcal{Z}$, $\mathcal{P}$ is activated to initialize as follows:
  let $(pk_p, sk_p) := \mathsf{Gen}(1^{\lambda})$
  let $txpool$ be an empty FIFO buffer
  let $chain := B_0$, where B_0 is the genesis block

/ * Receiving a longer chain * /
Upon receiving $chain'$ for the first time, the (online) $\mathcal{P}$ proceeds as:
  assert $|chain'| > |chain|$ and $\mathsf{validateChain}(chain') = 1$;
  let $chain := chain'$ and **broadcast** $chain$

/ * Receiving transactions * /
Upon receiving transactions(d') from $\mathcal{Z}$ (or other nodes) for the first time, the (online) $\mathcal{P}$ proceeds as:
  let $txpool.enqueue(d')$ and **broadcast** d'

/ * Main procedure * /
for each slot $sl' \in \{1, 2, \dots\}$, the (online) $\mathcal{P}$ proceeds as:
 if $\mathsf{eligible}(\mathcal{P}, sl') = 1$:
  let $d' := txpool.dequeue()$
  let $(header, d) := \mathsf{Head}(chain)$
  let $header' := (sl', st', G(d'), \pi')$, where $st' := H(header)$ and π' is the output of $\mathcal{P}$ (the signature or the nonce)
  let $chain := chain \| (header', d')$ and **broadcast** $chain$
 output extract($chain$) to $\mathcal{Z}$, where extract outputs an ordered list of each block in $chain$

Figure 8. Immutable Blockchain Protocol

D IDEAL IMMUTABLE BLOCKCHAIN PROTOCOL

We present the corresponding ideal functionality $\mathcal{F}'_{tree}$ (Figure 9) and the ideal immutable protocol Π'_{ideal} (Figure 10) for Γ', by pruning the redaction operations from $\mathcal{F}_{tree}$ (c.f. Figure 3) and Π_{ideal} (c.f. Figure 4), respectively.

$\mathcal{F}'_{tree}(p, p')$
On init: tree := genesis, time(genesis) := 0
On receive leader$(\mathcal{P}, t)$ from $\mathcal{A}$ or internally:
  if $\Gamma[\mathcal{P}, t]$ has not been set, let $\Gamma[\mathcal{P}, t] = \begin{cases} 1 & \text{with probability } \phi(p) \\ 0 & \text{otherwise} \end{cases}$
  return $\Gamma[\mathcal{P}, t]$
On receive extend(chain, B) from honest party $\mathcal{P}$:
  let t be the current time
  assert chain $\in$ tree, chain$\|$B $\notin$ tree, and leader$(\mathcal{P}, t)$ outputs 1
  append B to chain in tree, record time(chain$\|$B) := t
  return "succ"
On receive extend(chain, B, t') from corrupt party $\mathcal{P}^*$:
  let t be the current time
  assert chain $\in$ tree, chain$\|$B $\notin$ tree, leader$(\mathcal{P}, t)$ outputs 1, and time(chain) $< t' < t$
  append B to chain in tree, record time(chain$\|$B) := t'
  return "succ"
On receive verify(chain) from $\mathcal{P}$: return (chain $\in$ tree)

Figure 9. Ideal functionality $\mathcal{F}'_{tree}$

Ideal Protocol Π'_{ideal}
On init : chain := genesis
On receive chain$'$:
Assert $|\text{chain}'| > |\text{chain}|$ and $\mathcal{F}'_{tree}$.verify(chain$'$) = 1
For every slot:
–receive input B from $\mathcal{Z}$
–if $\mathcal{F}_{tree}$.extend(chain, B) outputs "succ", then let chain := chain$\|$B and broadcast chain
–output chain to $\mathcal{Z}$

Figure 10. Ideal Blockchain Protocol

E SECURITY PROOF OF THEOREM 4.3

Redactable common prefix. Assume that there exists B_j^* in the prefix of $\mathsf{chain}^{t'}_{\mathcal{P}_{i'}}(\mathsf{view})$ but not in the prefix of $\mathsf{chain}^{t}_{\mathcal{P}_i}(\mathsf{view})$ consisting of the first $|\mathsf{chain}^t_{\mathcal{P}_i}(\mathsf{view})| - k_0$ records, where $t \leq t'$, and a party $\mathcal{P}_i$ is honest at t and a party $\mathcal{P}_{i'}$ is honest at t' in view, which means B_j is redacted with B_j^* in $\mathsf{chain}^{t'}_{\mathcal{P}_{i'}}(\mathsf{view})$ but not in $\mathsf{chain}^{t}_{\mathcal{P}_i}(\mathsf{view})$. Then it must be the case that the party $\mathcal{P}_{i'}$ receives enough votes (more than the number of corrupt committee members) for B_j^* according to the ideal protocol specification. Therefore, the redaction policy $\mathcal{RP}$ is satisfied, and we conclude Π_{ideal} satisfies the k_0-redactable common prefix.

Chain quality. If an honest block B_j is replaced with a malicious block B_j^* (e.g., containing illegal or harmful data), the adversary $\mathcal{A}$ can increase the proportion of adversarial blocks in chain and finally break the chain quality property. However, according to the ideal protocol specification, an edited block can only be adopted

when the votes are more than the number of adversarial committee members. Since only those adversarial committee members would vote for the malicious block B_j^*, chain cannot be redacted. Therefore, we conclude Π_{ideal} satisfies the (k_0, μ)-chain quality.

Chain growth. Note that any edit operation would not alter the length of chain, since it is not possible to remove any blocks from chain according to the ideal protocol specification. Moreover, the new block issue process in current time slot is not influenced by votes for any edit request. No matter whether a party $\mathcal{P}$ has received enough votes, $\mathcal{P}$ always extends chain at time slot t as long as $\text{leader}(\mathcal{P}, t) = 1$. Therefore, we conclude Π_{ideal} satisfies the τ-chain growth. □

F SECURITY PROOF OF THEOREM 4.4

Consider a p.p.t. adversary $\mathcal{A}$ in the real-world protocol Γ. We construct the simulator $\mathcal{S}$ in the ideal protocol Π_{ideal} as follows:

(1) At the beginning of the protocol execution, $\mathcal{S}$ generates public/secret key pair $(pk_{\mathcal{P}}, sk_{\mathcal{P}})$ for each honest party $\mathcal{P}$, and stores the party $\mathcal{P}$ and public key $pk_{\mathcal{P}}$ mapping.

(2) For the leader selection process, we consider two common cases.
- The leader selection function eligible is modeled as the random oracle $H(\cdot)$. Whenever $\mathcal{A}$ sends a hash query $H(\mathcal{P}, t)$, $\mathcal{S}$ checks whether this query has been asked before and returns the same answer as before if so. Otherwise, $\mathcal{S}$ checks whether the identifier $\mathcal{P}$ corresponds to this protocol instance. If not, $\mathcal{S}$ samples a random number of the length $|H(\cdot)|$ and returns it to $\mathcal{A}$. Else if the check succeeds, $\mathcal{S}$ calls $b \leftarrow \mathcal{F}_{tree}.\text{leader}(\mathcal{P}, t)$, and returns b.
- The random oracle is replaced with normal function such as $\text{PRF}_k(\cdot)$. In this case, $\text{PRF}_k(\cdot)$ is used by both $\mathcal{S}$ and $\mathcal{A}$. Most of the simulation proof is identical to the random oracle case presented above, except that when $\mathcal{S}$ learns k from $\mathcal{F}_{tree}$, it simply gives k to $\mathcal{A}$, and $\mathcal{S}$ no longer needs to simulate random oracle queries for $\mathcal{A}$.

(3) $\mathcal{S}$ keeps track of the real-world *chain* for every honest party $\mathcal{P}_i$. Whenever it sends *chain* to $\mathcal{A}$ on behalf of $\mathcal{P}_i$, it updates this state for $\mathcal{P}_i$. Whenever $\mathcal{A}$ sends *chain* to honest party $\mathcal{P}_i$, $\mathcal{S}$ checks the simulation validity of *chain*. If it is valid and moreover *chain* is longer than the current real-world chain for $\mathcal{P}_i$, $\mathcal{S}$ also saves *chain* as the new real-world *chain* for $\mathcal{P}_i$.

(4) Whenever an honest party $\mathcal{P}$ sends chain to $\mathcal{S}$, $\mathcal{S}$ looks up the current real-world state *chain* for $\mathcal{P}$.
- If the editing pool $\mathcal{EP}$ is empty, $\mathcal{S}$ computes a new *chain*$'$ using the real-world algorithm. Specifically, let *sl* be the current slot, and if $\text{eligible}(\mathcal{P}, sl) = 1$, then $\mathcal{S}$ sets $B := (header', d')$ with $header' = (sl, st', G(d'), ib', \pi')$ such that $st' = H(header)$ and π' is the output of $\mathcal{P}$ (the signature for $\text{Head}(chain) = (header, d)$ or the nonce). Finally, $\mathcal{S}$ sets $chain' := chain \| B$ and sends *chain*$'$ to $\mathcal{A}$.
- If the editing pool $\mathcal{EP}$ is not empty (e.g., one candidate edited block B_j^* for B_j is included in $\mathcal{EP}$), $\mathcal{S}$ starts to collect the votes for B_j^* and simulates the vote process using the real-world algorithm. Specifically, for any party $\mathcal{P}_i$ who sends the candidate B_j^* to $\mathcal{S}$ in *sl*, if $\text{Cmt}(chain, \lfloor sl/w \rfloor * w, \mathcal{P}_i, para)$ returns $(c_i, proof_i)$, $\mathcal{S}$ votes for B_j^* in the name of $\mathcal{P}_i$ by computing $v_i = \text{Sign}(sk_i, H(B_j^*))$, and then sends $(c_i, proof_i, v_i)$ to $\mathcal{A}$. If $\mathcal{S}$ receives votes for B_j^*, $\mathcal{S}$ computes $(asig, PROOF)$ for B_j^* by the aggregation of v_i and $(c_i, proof_i)$. If $\text{eligible}(\mathcal{P}, sl') = 1$, $\mathcal{S}$ sets $d' := d' \| asig \| PROOF$ and $B := (header', d')$ with $header' = \{sl', st', G(d'), ib', \pi'\}$, such that $st' = H(header)$ and π' is the output of $\mathcal{P}$ (the signature for $\text{Head}(chain) = (header, d)$ or the nonce). Finally, $\mathcal{S}$ sets $chain' := chain \| B$ and sends *chain*$'$ to $\mathcal{A}$.

(5) Whenever $\mathcal{A}$ sends a message *chain* to an honest party $\mathcal{P}$, $\mathcal{S}$ intercepts the message and checks the validity of *chain* by executing the real-world protocol's checks (i.e., validateChain(.)). If the checks do not pass, $\mathcal{S}$ ignores the message. Otherwise,
- For the candidate edited block B_j^*, $\mathcal{S}$ abort outputting vote-failure if $\mathcal{RP}(chain, B_j^*, sl) = 1$ for some slot *sl* however $\mathcal{S}$ has never received enough votes for B_j^*.
- Else, let $\text{chain} := \text{extract}(chain)$, and let $\text{chain}[: l]$ be the longest prefix of chain such that $\mathcal{F}_{tree}.\text{verify}(\text{chain}[: l]) = 1$. If any block in $chain[l+1:]$ is signed by an honest party $\mathcal{P}$, $\mathcal{S}$ aborts outputting sig-failure. Else, for each $l' \in [l+1, |\text{chain}|]$, $\mathcal{S}$ calls $\mathcal{F}_{tree}.\text{extend}(\text{chain}[: l'-1], \text{chain}[l'], t')$ acting as a corrupted stakeholder $\mathcal{P}^*$, where $t' = Time$. Then $\mathcal{S}$ forwards *chain* to $\mathcal{P}$.

Lemma F.1. *If the signature scheme* SIG *is EUF-CMA secure and the hash function H is collision-resistant, the simulated execution never aborts with* sig-failure *except with negligible probability.*

Proof. Note that the adversary $\mathcal{A}$ cannot produce a malicious block $\widetilde{B_j^*}$ such that $H(\widetilde{B_j^*}) = H(B_j^*)$ for the candidate edited block B_j^*, since the hash function H is collision-resistant. Then, if sig-failure ever happens, the adversary $\mathcal{A}$ must have forged a signature on a new message that $\mathcal{S}$ never signed. Thus, we can immediately construct a reduction that breaks the EUF-CMA security of the underlying signature scheme SIG. Specifically, $\mathcal{S}$ simulates for $\mathcal{A}$ the protocol executing just as the above specification, and guesses a random party $\mathcal{P}_i$ whose signature security is broken. $\mathcal{S}$ generates the public/secret key pair for all other parties and produces the corresponding signatures. $\mathcal{S}$ also calls the signing oracle to generate signatures for $\mathcal{P}_i$. Eventually, if $\mathcal{A}$ outputs a valid signature σ and σ has never been previously output by the signing oracle, σ can be used as a forgery and EUF-CMA security of SIG is broken. □

Lemma F.2. *If the aggregate signature scheme* ASIG *is unforgeable and the function* Cmt *ensures the fraction (in terms of computational power or stake) of honest users in the committee is at least η, the simulated execution never aborts with* vote-failure *except with negligible probability.*

Proof. If vote-failure ever happens, the adversary $\mathcal{S}$ must have forged an aggregate signature *asig* on the individual messages in the name of the ξ parties, among which there is at least one honest stakeholder. Then we can construct a reduction that breaks the security of the underlying aggregate signature scheme ASIG. Specifically, $\mathcal{S}$ simulates the protocol executing for $\mathcal{A}$ as the above specification, and guesses a random party $\mathcal{P}_i$ as the honest party among the ξ parties. We denote by (pk^*, sk^*) the public/secret key pair of $\mathcal{P}_i$. $\mathcal{S}$ generates the public/secret key pair for all other parties and produces the corresponding signatures. $\mathcal{S}$ also calls the signing oracle $\text{Sign}(sk^*, .)$ to generate any signature for $\mathcal{P}_i$ as specified in the

security experiment. Eventually, if $\mathcal{A}$ outputs a valid aggregate signature *asig* on the message set $M = \{m^*, m_1, ..., m_{n-1}\}$ under the public key set $\{pk^*, pk_1, ..., pk_{n-1}\}$ and m^* has never been queried to the signing oracle $\text{Sign}(sk^*, .)$, where $n = \xi$, then *asig* can be used as a forgery and the security of ASIG is broken. □

Conditioned on the fact that all of the above failure events do not happen, the simulated execution is identically distributed as the real-world execution from the perspective of $\mathcal{Z}$. We thus complete the proof of theorem. □

G EXTENSION FOR MULTIPLE REDACTIONS

We extend the redactable protocol of Figure 1 to accommodate multiple redactions for each block. Intuitively, each redaction of one block must contain the entire history of previous redactions of that block, and can only be approved if all previous redactions (including the current one) are approved. In this extension, the history information is stored in the initial state component *ib*. We now sketch the main protocol changes.

Proposing an edit. To propose a redaction for block $B_j = (sl_j, st_j, G(d_j), ib_j, \pi_j, d_j)$, the user replaces d_j with the new data d_j^* and replaces ib_j with $ib_j^* = ib_j || G(st_j, d_j)$ if $ib_j \neq G(st_j, d_j)$. It then generates a candidate block $B_j^* = (sl_j, st_j, G(d_j^*), ib_j^*, \pi_j, d_j^*)$. Note that, if B_j has never been redacted before, then $ib_j = G(st_j, d_j)$ and thus $ib_j^* = G(st_j, d_j)$.

Valid Blocks. To validate a block, the users run the validateBlockExt algorithm (Algorithm 8). Intuitively, the validateBlockExt algorithm performs the same operations as the validateBlock algorithm (Algorithm 1), except that it consider the case where the block can be redacted multiple times. Note that *ib* stores the history information of the previous redactions, and thus can be parsed as $ib = ib^{(1)}||...||ib^{(l)}$ if the block has been redacted l times, where $ib^{(1)}$ denotes the original state information of the unredacted block version.

Algorithm 8 Extended block validation algorithm validateBlockExt(B)

Parse $B = (sl, st, G(d), ib, \pi, d)$;
Parse $ib = ib^{(1)}||...||ib^{(l)}$, where $ib^{(i)} \in \{0,1\}^* \ \forall i \in [l]$;
Validate data d, **if** invalid **return** 0;
Validate the leader, **if** invalid **return** 0;
Validate data π, **if** invalid **return** 0;
else return 1;

Valid Blockchains. To validate a chain, the users run the validateChainExt algorithm (Algorithm 9). The only difference from the original Algorithm 2 is that now $ib = ib^{(1)}||...||ib^{(l)}$ where $ib^{(1)}$ denotes the original state information of the unredacted block version.

Valid Candidate Editing Blocks. To validate a candidate editing block, the users run validateCandExt algorithm (Algorithm 10). If a block B_j has been redacted more than once, then validation of a candidate block B_j^* should account for the previous redactions. That is, the proof of each redaction must exist in the chain.

Algorithm 9 Extended chain validation algorithm validateChainExt(*chain*)

Parse $chain = (B_1, \cdots, B_m)$ and set $j = m$;
while $j \geq 2$ **do**
  parse $B_j = (sl_j, st_j, G(d_j), ib_j, \pi_j, d_j)$;
  parse $B_{j-1} = (sl_{j-1}, st_{j-1}, G(d_{j-1}), ib_{j-1}, \pi_{j-1}, d_{j-1})$;
  Parse $ib_j = ib_j^{(1)}||...||ib_j^{(l)}$, where $ib_j^{(i)} \in \{0,1\}^*$;
  Parse $ib_{j-1} = ib_{j-1}^{(1)}||...||ib_{j-1}^{(l')}$, where $ib_{j-1}^{(i)} \in \{0,1\}^*$;
  if Γ'.validateBlock$(B_j) = 0$ **then return** 0;
  else if $st_j = H(sl_{j-1}, st_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1}, \pi_{j-1})$,
    then $j = j - 1$;
  else if $st_j = H(sl_{j-1}, st_{j-1}, ib_{j-1}^{(1)}, ib_{j-1}^{(1)}, \pi_{j-1}) \wedge$
    $\mathcal{RP}(chain, B_{j-1}, sl_{j-1}) = 1$, **then** $j = j - 1$;
  else return 0.
end while
return Γ'.validateBlockExt(B_j).

Algorithm 10 Extended candidate block validation algorithm validateCandExt(*chain*, B_j^*)

Parse $B_j^* = (sl_j, st_j, G(d_j^*), ib_j^*, \pi_j, d_j^*)$;;
Parse $ib_j = ib_j^{(1)}||...||ib_j^{(l)}$, where $ib_j^{(i)} \in \{0,1\}^* \ \forall i \in [l]$;
if Γ'.validateBlock$(B_j^*) = 0$ **then return** 0;
Parse $B_{j-1} = (sl_{j-1}, st_{j-1}, G(d_{j-1}), ib_{j-1}, \pi_{j-1}, d_{j-1})$;
Parse $ib_{j-1} = ib_{j-1}^{(1)}||...||ib_{j-1}^{(l')}$, where $ib_{j-1}^{(i)} \in \{0,1\}^* \ \forall i \in [l']$;
Parse $B_{j+1} = (sl_{j+1}, st_{j+1}, G(d_{j+1}), ib_{j+1}, \pi_{j+1}, d_{j+1})$;
if $st_j \neq H(sl_{j-1}, st_{j-1}, ib_{j-1}^{(1)}, ib_{j-1}^{(1)}, \pi_{j-1})$
  or $st_{j+1} \neq H(sl_j, st_j, ib_j^{(1)}, ib_j^{(1)}, \pi_{j-1})$, **then return** 0;
for $i \in \{2, ..., l\}$ **do**
  if there is no valid $(asig, PROOF)$ for hash of the candidate block
    $H(sl_j, st_j, ib_j^{(i)}, ib_j^{(1)}||...||ib_j^{(i-1)})$ in the chain, **then return 0**.
end for
return 1.