Redactable Proof-of-Stake Blockchain with Fast Confirmation

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

Xinyu Li
xinyu2016@iscas.ac.cn
Institute of Software, Chinese Academy of Sciences

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

Bingyong Guo
guobingyong@tca.iscas.ac.cn
Institute of Software, Chinese Academy of Sciences

Han Feng
fenghan@tca.iscas.ac.cn
Institute of Software, Chinese Academy of Sciences

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

ABSTRACT

Blockchain technologies have received a considerable amount of attention, and immutability is essential property of blockchain which is paramount to applications such as cryptocurrency. However, “Right to be Fogotten" has been imposed in new European Union’s General Data Protection Regulation, making legally incompatible with immutalbe blockchains. Moveover, illicit data stored in immutable blockchain poses numerous challenge for law enforcement agencies such as Interpol. Therefore, it is imperative (even legally required) to design efficient redactable blockchain protocols in a controlled way.

In this paper, we present a redactable proof-of-stake blockchain protocol in the permissionless setting with fast confirmation. Our protocol offers public verifiability for redactable chains, and to prevent an adversary from targeted attack, also uses a verifiable random function to randomly select voters for redaction on different slots in a private and non-interactive way. Compared to previous solutions in permissionless setting, our redaction operation can be completed quickly, even only within one block in synchronous network, which is desirable for redacting harmful or sensitive data. Moreover, our protocol is compatible with most current proof-of-stake blockchains requiring only minimal changes. Furthermore, using simulation techniques, we prove that our protocol can achieve the security property of redactable common prefix, chain quality, and chain growth. Finally, we implement our protocol and provide experimental results showing that compared to immutable blockchain, the overhead incurred for different numbers of redactions in the chain is minimal.

KEYWORDS

Blockchain; Proof-of-Stake; Redactable Blockchain

1 INTRODUCTION

Blockchain protocols have been gaining increasing popularity and acceptance by a wider community, triggered by the first large-scale application of blockchains, i.e., the cryptocurrency Bitcoin [35]. In a nutshell, a blockchain is a *decentralized*, *public*, *immutable* and *ordered* ledger of records, which is created by establishing consensus among the chain’s participants. The consensus component can be achieved in a number of ways. The most popular is using proof-of-work such as Bitcoin [22, 35, 40], while proof-of-stake is emerging as one of the most promising alternative, since it does not rely on expensive hardware using vast amounts of electricity to compute mathematical puzzles as Bitcoin. In a proof-of-stake blockchain protocol [9, 16, 17, 30], roughly speaking, participants randomly elect one party to produce the next block by running a “leader election" process with probability proportional to their current stake (a virtual resource) held on blockchain.

Immutability of blockchain is paramount to applications such as cryptocurrency and payments, due to the fact that it ensures the history of payment transactions cannot be modified. However, with the adoption of the new European Union’s General Data Protection Regulation (GDPR) [3] in May 2018, it is no longer legally compatible with current immutable blockchains such as Bitcoin and Ethereum [2] to record personal data, since GDPR imposes the “Right to be Forgotten" (also known as Data Erasure) as a key Data Subject Right [28]. Moreover, an immutable ledger is not appropriate for some new applications [13] that are being envisaged for the blockchain such as government and public records [4, 21] and social media [1, 6]. The data stored on the chain may be illegal, harmful or sensitive, since the malicious user can abuse the ability of blockchain to post arbitrary transaction messages and moreover it is infeasible to filter all transaction data. If these illicit data contents cannot be removed from the blockchains, they may affect the life of people forever and further hinder future of the blockchains technology. For instance, Bitcoin blockchain contains child sexual abuse images [32], leaked private keys [39] and materials that infringe on intellectual rights [27]. More worse, immutability of blockchains facilitates illicit activities of international criminal groups, and brings the numerous challenges for law enforcement agencies such as Interpol [41]. In addition, smart contracts may not patch vulnerabilities if the blockchain is immutable, for example, 3,641,694 Ethers (worth of about 79 million of US dollars) are stolen due to the flaws of Ethereum and DAO contract [29], but vulnerabilities have to be patched by deploying a hard fork (i.e., a manual intervention operation performed by Ethereum developers).

To mitigate this problem, there must be a way to redact data content of blockchain in specific and exceptional circumstances, and redaction should be performed only under strict constraints, satisfying full transparency and accountability. In addition, the fast confirmation of redaction is imperative for some applications. In aforementioned examples, harmful or sensitive data should be redacted promptly, since otherwise the consequences are huge and even it is harmful for social security. If a redaction on social media rumors cannot be confirmed until one week later, it may be too late to stop irreparable damages.

1.1 Related Work

A straightforward approach to globally erasing or editing previously included data from a blockchain is to produce a hard fork and develop a new blockchain from the edited block. However, it requires a strong *off-chain* consensus among participants, which is notoriously difficult to achieve. To address this challenge, Ateniese et al. [8] firstly proposed the notion of redacting a blockchain. They use a chameleon hash function [12] to compute hash pointer, when redacting a block, a collision for the chameleon hash function can be computed by trusted entities with access to the chameleon trapdoor key. By this way, the block data can be modified while maintaining the chain consistency, and this solution has recently been commercially adopted by a consultancy company Accenture [7][23]. Recently, in order to support fine-grained and controlled redaction of blockchain, Derler et al. [18] 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. However, their solutions[8][18] using chameleon hash are rather limited in a permissioned setting. In permissionless blockchains like Bitcoin, users can join and leave the system at any time, and their solutions will suffer from scalability issues when sharing the trapdoor key among some miners and computing a collision for the chameleon hash function by a multi-party computation protocol.

Puddu et al. [38] 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 unencrypted version of the 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 running a multi-party computation protocol, and finally decrypt the appropriate version of the transaction as a new active transaction. However, their solution is still not suitable for permissionless setting. Concretely, the malicious users who establish redaction policy can escape redaction, or even break the stability of transactions by the affection among transactions. Moreover, μ chain also faces scalability problem when reconstructing decryption keys by the multi-party computation protocol.

Recently, Deuber et al. [19] proposed the first redactable blockchain protocol in the permissionless setting, which does not rely on heavy cryptographic primitives or additional trust assumption. Once a redaction requirement is proposed by any user, the protocol starts a consensus-based voting period, and only after obtaining enough votes for approving the redaction, the edition is really performed on the blockchain. The protocol offers public verifiability and accountability, that is, each user can verify whether a redaction on the blockchain is approved by checking the number of votes on the chain. Their solution is very elegant, however, the new joined user has to check all the blocks within the voting period to verify a redaction on the blockchain. Moreover, the voting period is very long, for example, 1024 consecutive blocks are required in their Bitcoin instantiation, which also means that it will take almost 7 days to confirm and publish a redaction block. Nevertheless, in practice, it is inefficient to redact harmful or sensitive data after such a long time, and it is also difficult to let new joined user in the system maintain these redactions.

1.2 Our Contributions

In this work, our overall goal is to propose a redactable proof-of-stake blockchain protocol in the permissionless setting with **fast confirmation**. More specifically, our technical contributions are threefold.

Redactable Proof-of-Stake Blockchain Protocol. We propose an approach to make the proof-of-stake blockchain redactable. On a high level, any stakeholder can propose a candidate edited block B_j^* for B_j in the chain *chain*, and only committee members (in the new slot *sl* of *chain*) can vote for B_j^*; if votes are approved by the editing policy (e.g., voted by the majority), the leader of the slot *sl* adds these votes and corresponding proofs to its block data collected and proposes a new block; and finally B_j is replaced by B_j^*. Specifically, our protocol has the following features.

- The redaction operation can be completed quickly. If the network is synchronous, the voting period is only within one block, and even in semi-synchronous or asynchronous network, the proposed redaction can also be performed after several blocks. Moreover, to validate an edited block, users can find all evidence only from one block in the chain.
- Whether a certain stakeholder has right to vote for redaction is decided via a private test that is executed locally using a verifiable random function (VRF) on a random seed and the new slot of the chain. This means that every stakeholder can independently determine if they are chosen to be on the voting committee, by computing a VRF with their own secret key, which prevents an adversary from targeting voting committee members. Moreover, stakeholders obtain voting rights in proportion to their stakes in the system, which means the more stakes owned by a user, the more voting power he or she has.
- Our protocol offers accountability for redaction, where any edited block in the chain is publicly verified. Moreover, multiple redactions per block can be performed throughout the run of the protocol.
- The design of our protocol is compatible with current proof-of-stake blockchain such as Ouroboros[9, 17, 30], NXT[15], PPCoin[31], and Snow White[16], i.e., it can be implemented right now and requires only minimal changes to the current blockchain, block, or transaction structures. Our redaction approach considers all the cases of synchronous, semi-synchronous, and asynchronous network. We believe compatibility is an important feature that must be preserved.

Security Analysis. We provide formal security definition of redactable blockchain along the lines of the seminal papers of Garay et al. [24] and Pass et al. [36]. In order to accommodate the edit operation, we take inspiration from Deuber's elegant work [19], and give an extended definition called redactable common prefix. Essentially, redactable common prefix considers the affect of edited data and requires that if the property of the common prefix is violated, it must be the case that there exist edited blocks satisfying the redaction policy $\mathcal{RP}$. Then we prove that our redactable proof-of-stake blockchain protocol satisfies redactable common prefix, chain quality and chain growth. Our proof relies on simulation techniques —

more specifically, first considers an idealized functionality $\mathcal{F}_{tree}$ that keeps track of all valid chains at any moment of time, 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.

Performance Evaluation. We develop a proof-of-concept implementation of our redaction approach on JD Chain [5], evaluating the overhead of adding our redaction mechanism. The results show that compared to immutable blockchain, the overhead incurred for different numbers of redactions in the chain is minimal. Moreover, independent of the chain size, there is a nearly constant overhead of about 14 minutes to complete one redaction (from proposing a redaction request to confirming this redeaction). In addition, all signatures of voting for an edited block are aggregated a multi-signature, which drastically reduces the communication complexity for proof-of-stake blockchain.

2 PRELIMINARIES

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

2.1 Verifiable Random Functions

The concept of verifiable random functions is introduced by Micali et al.[34]. Informally, it is a pseudo-random function that provides publicly verifiable proofs of its outputs' correctness.

Definition 2.1 (Verifiable Random Functions)[20]. A function family $F_{(\cdot)}(\cdot) : \{0, 1\}^l \to \{0, 1\}^{l_{VRF}}$ 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 runs for a total of $s(\gamma)$ steps when its first input is 1^γ, and does not query the oracle on x,

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

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.

2.2 Signature Scheme

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

To define security [26], we consider the following game between an adversary $\mathcal{A}$ and a challenger.

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

Definition 2.2 (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

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

2.3 Multi-Signature Scheme

A multi-signature scheme [10, 33] is a protocol that enables the n signers to jointly generate a short signature *msig* on m so that *msig* convinces a verifier that all n parties signed m.

A multi-signature scheme is defined as algorithms Pg, Kg, Sign, KAg, and Vf. The system parameters $par \leftarrow \mathsf{Pg}$ are generated by a trusted party. Each signer generates a pair of key $(pk, sk) \xleftarrow{\$} \mathsf{Kg}(par)$, and signers can collectively sign a message m by each running the interactive algorithm $\mathsf{Sign}(par, PK, sk, m)$, where PK is the set of the public keys of the signers, and sk is the signer's individual secret key. In the end, every signer will outputs a signature σ. Algorithm KAg outputs a single aggregate public key *apk* on inputs a set of public keys PK. A verifier check the validity of a signature σ on message m under an aggregate public key *apk* by calling the algorithm $\mathsf{Vf}(par, apk, m, \sigma)$ which outputs 1 if the signatures is valid and 0 otherwise.

A multi-signature scheme should satisfy completeness, which means that for any n, if we have $(pk_i, sk_i) \leftarrow \mathsf{Kg}(par)$ for $i = 1, \dots, n$, and for any message m, if all signers input $\mathsf{Sign}(par, sk_i, m)$, then they will output a signature σ such that $\mathsf{Vf}\left(par, \mathsf{KAg}(par, \{pk_i\}_{i=1}^n), m, \sigma\right) = 1$.

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

(1) Setup Phase. The challenger generates the parameters $par \leftarrow \mathsf{Pg}$ and a challenge key pair by calling $(pk^*, sk^*) \xleftarrow{\$} \mathsf{Kg}(par)$. It runs the adversary on the public key $\mathcal{A}(par, pk^*)$.
(2) Signing Phase. $\mathcal{A}$ can make signature queries on any message m for any set of signer public keys PK with $pk^* \in PK$ which means that it has access to oracle $O^{\mathsf{Sign}(par, \cdot, sk^*, \cdot)}$ that will simulate the honest signer interacting in a signing protocol with the other signers of PK to signer message m. Note that $\mathcal{A}$ is allowed to make any number of such queries concurrently.
(3) Forgery Phase. $\mathcal{A}$ outputs a multi-signature forgery σ, a message m^*, and a set of public keys PK. The adversary wins if $pk^* \in PK$, $\mathcal{A}$ made no signing queries on m^*, and $\mathsf{Vf}(par, \mathsf{KAg}(par, PK), m^*, \sigma) = 1$.

Definition 2.3 (Unforgeability). We say that a multi-signature scheme MSIG is *unforgeable*, if for all adversaries $\mathcal{A}$, there exists a negligible function negl(par) such that

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

3 FORMAL ABSTRACTION OF BLOCKCHAIN

In this section, we define the formal abstraction and security properties of a blockchain. Our definitions are based on the approach of Garay et al.[24] and Pass et al.[36][37].

3.1 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.

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 the environment $\mathcal{Z}$. The environment $\mathcal{Z}$ 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, the party i is removed from the protocol execution.

3.2 Security Properties of Blockchain

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.

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 3.1. (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 3.2. (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 3.3. (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:

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

4 REDACTING PROOF-OF-STAKE BLOCKCHAIN

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

4.1 Proof-of-Stake Blockchain Protocol

We recall basic definitions [17] of proof-of-stake blockchain. There are n stakeholders $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and each stakeholder $\mathcal{P}_i$ possesses s_i stake and 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 $B := (sl, st, d, \sigma)$, where $sl \in \{sl_1, \cdots, sl_R\}$ is the slot number, $st \in \{0, 1\}^{\lambda}$ is the hash of the previous block, $d \in \{0, 1\}^*$ is the block data, and σ is a signature on (sl, st, d) computed under the secret key of slot leader generating the block.

A valid blockchain *chain* relative to the genesis block B_0 is a sequence of blocks $B_1, \cdots, B_m$ associated with a strictly increasing sequence of slots, where B_0 contains auxiliary information and the list of stakeholders identified by their public-keys and their respective stakes $(pk_1, s_1), \ldots, (pk_n, s_n)$. We use $\mathsf{Head}(chain)$ to denote the head of $chain$ (i.e., the block B_m). In a basic proof-of-stake blockchain protocol, the users always update their current chain to the longest valid chain they have seen so far. Let $\mathsf{eligible}(\mathcal{P}_i, s_i, sl)$ be a function that determines whether a stakeholder $\mathcal{P}_i$ with the stake s_i is an eligible leader at the time slot sl, and $\mathcal{P}_i$ can create and broadcast a block at sl if $\mathsf{eligible}(\mathcal{P}_i, s_i, sl) = 1$, where the leader election can be achieved according to specific proof-of-stake blockchain protocol.

4.2 Overview of Redactable Proof-of-Stake Blockchain Protocol

We construct our redactable blockchain protocol Γ by modifying and extending the basic proof-of-stake blockchain protocol. We assume that the fraction of stakes held by honest users is above threshold h (a constant greater than $\frac{2}{3}$). First, a redaction policy is introduced to determine whether an edit to the blockchain should be approved or not.

Definition 4.1. (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^* is at least $\frac{2}{3} \cdot T$, where votes are embedded in a block B_r, $B_r \in chain[:-k]$, and T is a parameter that determines the expected number of stake in voting committee[1].

Next, in order to accommodate editable data, we extend the above block structure to be of the form $B := (sl, st, d, ib, \sigma)$. Specifically, if a blockchain $chain$ with $\mathsf{Head}(chain) = (sl, st, d, ib, \sigma)$ is updated to a new longer blockchain $chain' = chain \| B'$, the newly created block $B' = (sl', st', d', ib', \sigma')$ sets $st' = H(sl, G(st, d), ib, \sigma)$ and $ib' = G(st', d')$, where H and G are prescribed collision-resistent hash functions, σ' is a signature on $(sl', G(st', d'), ib')$ computed under the secret key of slot leader generating the block B'. Notice that in order to maintain the link relationships between an edited block and its neighbouring blocks, inspired by the work [19] we introduce ib to represent the initial and unedited state of block, i.e., $ib = G(st, d_0)$ if original block data is d_0 in the edited block $B = (sl, st, d, ib, \sigma)$.

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

(1) If a user wishes to propose an edit to block B_j in $chain$, he first parses $B_j = (sl_j, st_j, d_j, ib_j, \sigma_j)$, replaces d_j with the new data d_j^*, and then broadcasts the candidate block $B_j^* = (sl_j, st_j, d_j^*, ib_j, \sigma_j)$ to the network, where $d_j^* = \varepsilon$ if the user wants to remove all data from B_j.

(2) Upon receiving B_j^* from the network, every stakeholder $\mathcal{P}_i$ first validates whether B_j^* is a valid candidate editing block, and stores it in his own editing pool $\mathcal{EP}_i$ if it is. Notice that each candidate editing block in the pool $\mathcal{EP}$ has a period of validity t_p.

(3) For each new slot sl, the leader creates a block and broadcasts $chain$ in exactly the same manner as the basic proof-of-stake blockchain if his editing pool is empty. Otherwise, the leader collects and validates the votes on edited block B_j^* in his editing pool by using sub-protocol $\mathsf{collectVote}$ (Figure 3). If it holds and returns $(msig, PROOF)$, the leader replaces B_j with B_j^*, adds $(msig, PROOF)$ to the data d', creates a new block and broadcasts $chain$, where d' is the new block data collected.

(4) At the beginning of each new slot sl, every stakeholder $\mathcal{P}_i$ tries to update his own editing pool $\mathcal{EP}_i$. For every candidate editing block B_j^* in $\mathcal{EP}_i$, $\mathcal{P}_i$ first checks whether B_j^* has expired or not, and if it is, $\mathcal{P}_i$ removes B_j^* from $\mathcal{EP}_i$. Then $\mathcal{P}_i$ computes $\mathcal{RP}(chain, B_j^*, sl_j)$ to check whether B_j^* should be adopted in the chain, and if it outputs 1, $\mathcal{P}_i$ replaces B_j in $chain$ with B_j^* and removes B_j^* from $\mathcal{EP}_i$. Finally, for every remaining candidate editing block B_j^* in the $\mathcal{EP}_i$, $\mathcal{P}_i$ with stake s_i checks whether he has voting right for this block in current slot sl by using sub-protocol $\mathsf{checkVote}$ (Figure 2). If it holds, $\mathcal{P}_i$ broadcasts $(hash, \pi)$ and the signature sig on $H(B_j^*)$ with his own secret key sk_i.

Redactable proof-of-stake 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 $st_j \neq H(sl_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1}, \sigma_{j-1})$, $chain$ is valid only under the condition of $st_j = H(sl_{j-1}, ib_{j-1}, ib_{j-1}, \sigma_{j-1})$ and $\mathcal{RP}(chain, B_{j-1}, sl_{j-1}) = 1$.

4.3 Redactable Proof-of-Stake 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 $\mathsf{validateBlock}$ 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 σ with the public key pk of the leader. In particular, for an edited block, the signature σ is on the "old" state (sl, ib, ib). We say that B is a valid block iff $\mathsf{validateBlock}(B)$ outputs 1.

procedure validateBlock(B)
Parse $B = (sl, st, d, ib, \sigma)$;
Validate data d, **if** invalid **return** 0;
Validate the leader, **if** invalid **return** 0;
if the signature σ on $(sl, G(st, d), ib)$ or on (sl, ib, ib) is verified with pk, **then return** 1;
else return 0.

Algorithm 1: The block validation algorithm

Valid Blockchains. To validate a blockchain $chain$, the $\mathsf{validateChain}$ 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}

[1]In blockchain protocol, a transaction can be finally confirmed after k blocks, and the selection of T will be discussed in Section 4.4.

has been redacted (i.e., $st_j \neq H(sl_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1}, \sigma_{j-1})$), its check additionally depends on whether the redaction policy $\mathcal{RP}$ of the blockchain has been satisfied. We say that *chain* is a valid blockchain iff validateChain(*chain*) outputs 1.

procedure validateChain($chain$)

Parse $chain = (B_1, \cdots, B_m)$;
if $m = 1$ **then return** validateBlock(B_1);
otherwise, **return** 1 **if** for all $j \in [2..m]$, $B_j = (sl_j, st_j, d_j, ib_j, \sigma_j)$:
1. validateBlock(B_j) = 1;
2. $st_j = H(sl_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1}, \sigma_{j-1})$ **or**
3. $st_j = H(sl_{j-1}, ib_{j-1}, ib_{j-1}, \sigma_{j-1})$ and $\mathcal{RP}(chain, B_{j-1}, sl_{j-1}) = 1$

Algorithm 2: The blockchain validation algorithm

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 block 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., $st_{j+1} = H(sl_j, ib_j, ib_j, \sigma_j)$. We say that B_j^* is a valid candidate editing block iff validateCand(*chain*, B_j^*) outputs 1.

procedure validateCand(C, B_j^*)

Parse $B_j^* = (sl_j, st_j, d_j^*, ib_j, \sigma_j)$;
if validateBlock(B_j^*) = 0 **then return** 0;
Parse $B_{j-1} = (sl_{j-1}, st_{j-1}, d_{j-1}, ib_{j-1}, \sigma_{j-1})$;
Parse $B_{j+1} = (sl_{j+1}, st_{j+1}, d_{j+1}, ib_{j+1}, \sigma_{j+1})$;
if $st_j = H(sl_{j-1}, ib_{j-1}, ib_{j-1}, \sigma_{j-1})$ and $st_{j+1} = H(sl_j, ib_j, ib_j, \sigma_j)$
  then return 1;
else return 0.

Algorithm 3: The candidate block validation algorithm

We now present redactable proof-of-stake blockchain protocol Γ in Figure 1. Γ is parameterized by redaction policy $\mathcal{RP}$, corrupted stakes ratio ρ, and expected number of stakes in voters committee T, where $\rho = 1 - h < 1/3$. The subroutines checkVote and collectVote are used to check stakeholder's voting right and collect the votes, respectively.

Checking voting right. The subroutine checkVote (Figure 2) checks a stakeholder $\mathcal{P}_i$ (with secret key sk_i and stake s_i) whether having right to vote. Inspired by the idea of Algorand [25], checkVote uses VRFs to randomly select voters in a private and non-interactive way. Specifically, $\mathcal{P}_i$ computes $(hash, \pi) \leftarrow VRF_{sk_i}(seed\|sl)$ with his own secret key sk_i, where 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 S is total stakes in the system, T is the expected number of stakes in committee for voting, $C(s_i, q) = \frac{s_i!}{q!(s_i-q)!}$, 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*.

Redactable Proof-of-Stake Blockchain Protocol Γ

On input initialization request init() from $\mathcal{Z}$:
  Let $(pk, sk) := \text{Sig.Gen}(1^\lambda)$
  Let $chain$ be genesis block B_0, $chain := B_0$
On receive $chain'$:
  Assert $|chain'| > |chain|$ and validateChain($chain'$)= 1;
  Let $chain := chain'$ and broadcast $chain$
For every slot sl':

- On receive input transactions(d') from $\mathcal{Z}$:
  - If eligible($\mathcal{P}$, s, sl')= 1 and the editing pool $\mathcal{EP}$ is empty, where $\mathcal{P}$ is the current node with the stake s:
    Let $B := (sl', st', d', ib', \sigma')$, such that $st' = H(sl, G(st, d), ib, \sigma)$ and $\sigma' = \text{Sig.Sign}(sk_{\mathcal{P}}; sl', G(st', d'), ib')$ for Head($chain$) = (sl, st, d, ib, σ);
    Let $chain := chain \| B$ and broadcast $chain$
- On receive input transactions(d') from $\mathcal{Z}$:
  - If eligible($\mathcal{P}$, s, sl')= 1 and the editing pool $\mathcal{EP}$ is not empty, where $\mathcal{P}$ is the current node with the stake s:
    Collect the votes for every candidate editing block B_j^* by calling collectVote;
    Let $d' := d' \| msig \| PROOF$ and $chain[j] := B_j^*$ if collectVote returns $(B_j^*, msig, PROOF)$;
    Let $B := (sl', st', d', ib', \sigma')$, such that $st' = H(sl, G(st, d), ib, \sigma)$ and $\sigma' = \text{Sig.Sign}(sk_{\mathcal{P}}; sl', G(st', d'), ib')$ for Head($chain$) = (sl, st, d, ib, σ);
    Let $chain := chain \| B$ and broadcast $chain$
- On receive input edit(B_j^*) from $\mathcal{Z}$ and if the current node is $\mathcal{P}$:
    Assert B_j^* is a valid candidate editing block;
    Add B_j^* to the editing pool $\mathcal{EP}$ of $\mathcal{P}$ and remove expired blocks in $\mathcal{EP}$
- If the current node is $\mathcal{P}$:
    For every candidate editing block B_j^* in the editing pool $\mathcal{EP}$ of $\mathcal{P}$, let $chain[j] := B_j^*$ and remove B_j^* from $\mathcal{EP}$ if $\mathcal{RP}(chain, B_j^*, sl_j) = 1$;
    Check the voting right by calling checkVote;
    For every remaining candidate editing block B_j^* in the $\mathcal{EP}$ of $\mathcal{P}$, broadcast the voting information $(hash, \pi)$ and $sig = \text{Sig.Sign}(sk_{\mathcal{P}}; H(B_j^*))$ if checkVote returns $(hash, \pi)$
- Output extract($chain$) to $\mathcal{Z}$, where extract outputs an ordered list of each block in $chain$

Figure 1. Redactable Proof-of-Stake Blockchain Protocol Γ

subroutine checkVote(sl, sk_i, s_i, $seed$, T, S)

$(hash, \pi) \leftarrow VRF_{sk_i}(seed\|sl)$;
$p \leftarrow \frac{T}{S}$;
$c \leftarrow 0$;
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**
$c \leftarrow c + 1$
if $c \neq 0$ **then return** $(hash, \pi)$
else return 0.

Figure 2. Checking Voting Right

Collecting votes. The subroutine collectVote (Figure 3) collects and validates the votes. The collected votes are stored in *msgs* buffer. To validate a vote, it first verifies the signature on $H(B_j^*)$ under the public key of the voter, and then verifies a proof $(hash, \pi)$ to confirm the voting right of the voter, i.e., $\mathsf{VerifyVRF}_{pk}(hash, \pi, seed\|sl)$[2]. If the voter $\mathcal{P}_i$ was chosen c times (i.e., $\frac{hash}{2^{hashlen}}$ falls in the interval I^c), the number of votes from $\mathcal{P}_i$ is c as well. As soon as the number of votes collected is more than $\frac{2}{3} \cdot T$, the algorithm generates a multi-signature $msig$ on all these vote signatures SIG, aggregates corresponding proofs $PROOF$, and returns them, where multi-signature can reduces the communication complexity and storage overhead for proof-of-stake blockchain. If not enough votes are collected within the allocated τ_t time window, then the algorithm returns 0.

Observe that in a synchronous network, messages are delivered within a maximum network delay of Δ and we can set $\tau_t = \Delta$. While in partially synchronous or asynchronous network, we can not obtain such Δ. We firstly set $\tau_t = t$, and if the leader in this slot does not obtain enough votes of a candidate editing block because of network delay, then the block will be voted again in the next slot, where we set $\tau_t = 2t$. The time window will increase exponentially with slot until the candidate editing block expires. By this way, it is very likely that a candidate editing block will be approved eventually unless message delays grow faster than the time window indefinitely, which is unlikely in a real system.

```
subroutine collectVote(msgs, sl, seed, T, S, τ_t)
start ← Time();
votes ← 0;
SIG ← {};
PROOF ← {};
For every m ← msgs.next()
  if Time() > start + τ_t then return 0;
  else
    (hash, π, sig) ← m;
    if the signature sig on H(B_j^*) is verified
      then continue;
    if VerifyVRF_pk(hash, π, seed||sl) = 1
      then continue;
    p ← T/S;
    c ← 0;
    while hash/2^hashlen ∉ (Σ_{q=0}^{c} B(q; s_i, p), Σ_{q=0}^{c+1} B(q; s_i, p)) do
      c ← c + 1;
    votes = votes + c;
    SIG = SIG ∪ {sig};
    PROOF = PROOF ∪ {(hash, π)};
  if votes > 2/3 · T
   then compute multi-signature msig on SIG
   and return (B_j^*, msig, PROOF).
```

Figure 3. Collecting Votes

4.4 The Number of Voting Committee

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". Let S be the total number of stakes in the system (S 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

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

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:

$$\frac{T^K}{K!}e^{-T} \quad (1)$$

When we select the value of T, we want the number of honest committee members is more than $l_s \cdot T$, where $l_s \cdot T$ are some predetermined threshold. The condition is violated when the number of honest committee members is not more than $l_s \cdot T$. From formula (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 at least h ($h > \frac{2}{3}$). Thus, the probability that the condition is violated is given by the formula

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

F is a parameter which marks a negligible probability that the condition fails, and our experience sets $F = 5\times10^{-9}$. Our goal is to minimize T, while maintaining the probability that the condition fails to be at most F. If some value of T satisfies the condition with probability $1-F$, then any larger value of T also does for the same l_s 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 see if we can find a $l_s \in (\frac{2}{3}, 1]$ that satisfies the condition. If such l_s exists, then we decrease T and see if we also can find a good l_s. We continue this process until finding the optimal number of committee and corresponding threshold l_s. In this way, we can get Figure 4, plotting the expected committee size T satisfying the condition, 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 [25].

In the implementation of our system, we assume the fraction of honest stakes is $\frac{3}{4}$, so we select $T = 1000$ according to Figure 4. From the subroutine collectVote (Figure 3), a validate editing block is approved only after it obtains more than $\frac{2}{3}\cdot T$ votes.

We stress that the number of votes from malicious stakeholders cannot reach $\frac{2}{3}\cdot T$ with non-negligible probability. Specifically, when the size n of selected committee members satisfies $n > T$, the number of honest committee members is more than $\frac{2}{3}\cdot n$ with probability at least $1-F$ according to the above discussion, while the malicious committee members can only reach $\frac{2}{3}\cdot T$ unless $\frac{1}{3}\cdot n > \frac{2}{3}\cdot T$ (i.e., $n > 2T$), which occurs with a negligible probability since T

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

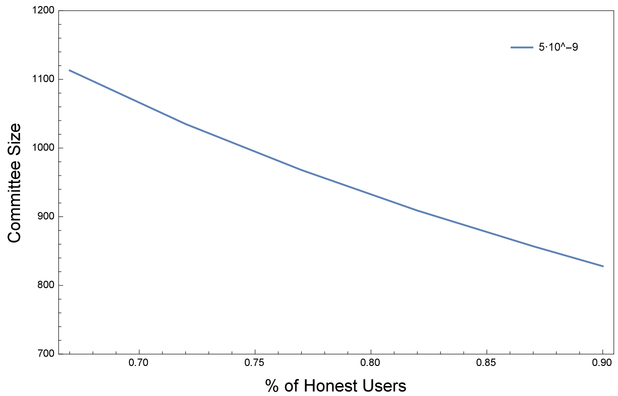


Figure 4. The x-axis specifies h, the stakes fraction of honest users. The committee size, T, is sufficient to limit the probability of violating safety to 5×10^{-9}.

is the expected value of the committee size following the binomial distribution. Similarly, when $n < T$, the malicious members can only obtain more than $\frac{2}{3} \cdot T$ votes unless $\rho' \cdot n > \frac{2}{3} \cdot T (i.e., \rho' > \frac{2}{3})$, where ρ' denotes the fraction of malicious committee members. This, however, only occurs with a negligible probability, since n cannot deviate from T too far as discussed above, that is, the fraction of malicious members cannot exceed $\frac{1}{3}$ too much. This result keeps consistent with that in Algorand [25].

5 SECURITY ANALYSIS

In this section, we analyze the security of redactable proof-of-stake blockchain protocol Γ as depicted in Figure 1. The security properties of redactable blockchain are same as that of basic blockchain, except for the common prefix property.

Redactable Common Prefix. We observe that redactable proof-of-stake 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 block 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 3.1.

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 consistency 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 editing policy $\mathcal{RP}$.

Let $\mathsf{redactprefix}^k(\mathsf{view}) = 1$ if for all times $t \leq t'$, 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 5.1. (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:

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

Essentially, Γ behaves just like the underlying immutable proof-of-stake blockchain protocol in Appendix A 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 proof-of-stake blockchain protocol under the redaction policy $\mathcal{EP}$.

Theorem 5.2. **(Security of Γ).** *Assume that the signature scheme* SIG *is EUF-CMA secure, the multi-signature scheme* MSIG *is unforgeable,* VRF *satisfies the properties of Definiton 2.1, and the underlying immutable blockchain protocol in Appendix A satisfies k_0-common prefix, (k_0, μ)-chain quality, and τ-chain growth. Then, redactable proof-of-stake blockchain protocol Γ satisfies the k_0-redactable common prefix, (k_0, μ)-chain quality, and τ-chain growth.*

Proof roadmap. We first consider a simple ideal-world protocol denoted 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. Then we show that the real-world protocol Γ securely emulates the ideal-world protocol Π_{ideal}. We prove the theorem in the following two subsections.

5.1 Security of Ideal Protocol Π_{ideal}

We first define an ideal functionality $\mathcal{F}_{tree}$ (Figure 5) and analyze an ideal-world protocol Π_{ideal} (Figure 6) parameterized with $\mathcal{F}_{tree}$.

The ideal functionality $\mathcal{F}_{tree}$ keeps track of the set (denoted 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 (or committee member, resp.) for every time step with probability $\phi(s, p)$ (or $\phi(s, p')$, resp.), where ϕ is a general function whose output is proportional to the stake 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 through the $\mathcal{F}_{tree}$.leader (or $\mathcal{F}_{tree}$.committee, resp.) query. Further, honest and corrupt parties can extend known chains with new block by calling $\mathcal{F}_{tree}$.extend, if they are elected as leader for a specific time step. Specifically, honest parties always extend chains in the current time, while corrupt 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 and redact the blockchain, if the votes are more than the number of corrupt 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.

$\mathcal{F}_{tree}(p, p')$
On init: tree := genesis, time(genesis) := 0
On receive leader($\mathcal{P}$, t) from $\mathcal{A}$ or internally:
  let s be the stake 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 $\Gamma[\mathcal{P}, t]$
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) = 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) = 1, and time(chain) $< t' < t$
  append B to chain in tree, record time(chain||B) := t'
  return "succ"
On receive committee($\mathcal{P}$, t) from $\mathcal{A}$ or internally:
  let s be the stake 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 $\Gamma'[\mathcal{P}, t]$
On receive redact(chain, i, B^*) from λ distinct parties $\mathcal{P}_j$:
  let t be the current time
  assert chain $\in$ tree and committee($\mathcal{P}_j$, t) = 1 for every $\mathcal{P}_j$
  assert λ is more than the number of corrupt parties $\mathcal{P}_j$ with
    committee($\mathcal{P}_j$, t) = 1
  redact chain[i] := B^* and return "succ"
On receive verify(chain) from $\mathcal{P}$: return (chain $\in$ tree)

Figure 5. Ideal Functionality $\mathcal{F}_{tree}$

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

Figure 6. Ideal Protocol Π_{ideal}

Theorem 5.3. **(Security of** Π_{ideal}**).** *If the underlying immutable ideal protocol in Appendix B 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.*

Proof. Note that if there is no edit in chain, then Π_{ideal} behaves exactly like the underlying immutable iedal protocol in Appendix B, and thus k_0-common prefix, (k_0, μ)-chain quality, and τ-chain growth can be preserved directly.

Redactalble common prefix. Assume that there exists B_j^* in the prefix of chain$^{t'}_{\mathcal{P}_{i'}}$(view) but not in the prefix of chain$^{t}_{\mathcal{P}_i}$(view) consisting of the first |chain$^{t}_{\mathcal{P}_i}$(view)| $- k_0$ records, where $t \le t'$, 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 chain$^{t'}_{\mathcal{P}_{i'}}$(view) but not in chain$^{t}_{\mathcal{P}_i}$(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. In case of an edit, the adversary $\mathcal{A}$ can increase the proportion of adversarial blocks in chain and finally break the chain quality property, if an honest block B_j is replaced with a malicious block B_j^* (e.g., containing illegal or harmful data). 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 within pre-defined time window, $\mathcal{P}$ always extends chain at time slot t as long as leader$(\mathcal{P}, t) = 1$. Therefore, we conclude Π_{ideal} satisfies the τ-chain growth. □

5.2 Real-world Emulates Ideal-world

So far, we have proved that the ideal-world protocol Π_{ideal} satisfies the k_0-redactable common prefix, (k_0, μ)-chain quality, and τ-chain growth. We next show that the real-world protocol Γ as depicted in Figure 1 emulates the ideal-world protocol Π_{ideal}, and thus Γ also satisfies the same three security properties.

Theorem 5.4. **(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:*

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

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

Proof. Consider some 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 $H(\mathcal{P}, s, t)$, $\mathcal{S}$ 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 $b \leftarrow \mathcal{F}_{tree}$.leader($\mathcal{P}, t$). If $b = 1$ (or $b = 0$, resp.), $\mathcal{S}$ picks h uniformly at random from $\{0,1\}^{|H(\cdot)|}$ with rejection sampling until h satisfies eligible = 1 (or eligible = 0, resp.), and then returns h.

- The random oracle is replaced with normal function such as $\mathsf{PRF}_k(\cdot)$. In this case, $\mathsf{PRF}_k(\cdot)$ 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 stakeholder $\mathcal{P}$ 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 eligible($\mathcal{P}$, s, sl) = 1, then $\mathcal{S}$ sets $B := (sl', st', d', ib', \sigma')$, such that $st' = H(sl, G(st, d), ib, \sigma)$ and $\sigma' = \mathsf{Sig.Sign}(sk_{\mathcal{P}}; sl', G(st', d'), ib')$ for Head(*chain*) $= (sl, st, d, ib, \sigma)$. 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 $\mathcal{EP}$), and eligible($\mathcal{P}$, s, sl) = 1, $\mathcal{S}$ starts to collect the votes for B_j^* and simulate the vote process using the real-world algorithm. Specifically, for any stakeholder $\mathcal{P}_i$ who sends the candidate B_j^* to $\mathcal{S}$ in the current slot *sl*, if checkVote($sl, sk_i, \cdot$) return $(hash_i, \pi_i)$, $\mathcal{S}$ votes for B_j^* in the name of $\mathcal{P}_i$ by computing the pair (v_i, c_i), where $v_i = \mathsf{Sig.Sign}(sk_i, H(B_j^*))$ and c_i is computed as in Figure 2, and then sends v_i to $\mathcal{A}$. If in the current slot $\mathcal{S}$ receives at least $\lambda+1$ votes for B_j^*, $\mathcal{S}$ computes $(msig, PROOF)$ for B_j^* by the aggregation of v_i and $(hash_i, \pi_i)$. Finally, $\mathcal{S}$ sets $d' := d' \| msig \| PROOF$, $B := (sl', st', d', ib', \sigma')$, such that $st' = H(sl, G(st, d), ib, \sigma)$ and $\sigma' = \mathsf{Sig.Sign}(sk_{\mathcal{P}}; sl', G(st', d'), ib')$ for Head(*chain*) $= (sl, st, d, ib, \sigma)$, sets $chain' := chain \| B$, and sends *chain*$'$ to $\mathcal{A}$.

(5) Whenever $\mathcal{A}$ sends a protocol message *chain* to an honest stakeholder $\mathcal{P}$, $\mathcal{S}$ intercepts the message and checks the validity of *chain* by running 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 chain := extract(*chain*), and let chain[: l] be the longest prefix of chain such that $\mathcal{F}_{tree}$.verify(chain[: l]) = 1. If any block in $chain[l+1:]$ is signed by an honest stakeholder $\mathcal{P}$, $\mathcal{S}$ aborts outputting sig-failure. Else, for each $l' \in [l+1, |\text{chain}|]$, $\mathcal{S}$ calls $\mathcal{F}_{tree}$.extend(chain[: $l'-1$], chain[l'], t') acting as the corrupt stakeholder $\mathcal{P}^*$, where t' = time(*chain*). Then $\mathcal{S}$ forwards *chain* to $\mathcal{P}$.

Lemma 5.5. *If the signature scheme* SIG *is EUF-CMA secure, the simulated execution never aborts with* sig-failure *except with negligible probability.*

Proof. Note that 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 running just as the above specification, and guesses a random stakeholder $\mathcal{P}_i$ 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 5.6. *If the multi-signature scheme* MSIG *is unforgeable,* VRF *satisfies the properties of Definiton 2.1, the simulated execution never aborts with* vote-failure *except with negligible probability.*

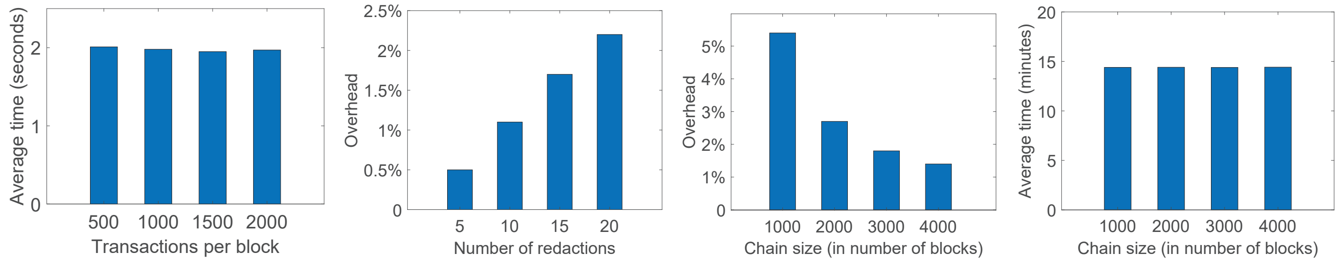
Proof. If vote-failure ever happens, the adversary $\mathcal{S}$ under static corruption must have forged a multi-signature *msig* on a message in the name of the $2/3 \cdot T$ stakeholders, among which there is at least one honest stakeholder. Then we can construct a reduction that breaks the security of the underlying multi-signature scheme MSIG. Specifically, $\mathcal{S}$ simulates the protocol running for $\mathcal{A}$ as the above specification, and guesses a random stakeholder $\mathcal{P}_i$ as the honest stakeholder among the $2/3 \cdot T$ stakeholders. $\mathcal{S}$ generates the public/secret key pair for all other parties and produces the corresponding signatures. $\mathcal{S}$ also calls the signing oracle of $\mathcal{P}_i$ for any signature to generate for $\mathcal{P}_i$ as specified in the security experiment. Eventually, if $\mathcal{A}$ outputs a valid multi-signature *msig* on some message m and m has never been queried to the singing oracle of $\mathcal{P}_i$, *msig* can be used as a forgery and the security of MSIG is broken.

For the adversary $\mathcal{A}$ under adaptive corruption, he can employ the ability of adaptive corruption during the voting process to vote for his adversarial request, which leads to vote-failure. If $\mathcal{A}$ can "presciently" ensure which user would become the member of the voting committee, he can adaptively corrupt and impersonate this user to vote for his request, such that the votes for the adversarial request exceed $\frac{2}{3} \cdot T$ and the edit request is adopted. 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 $1/2^{hashlen}$. □

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. □

6 IMPLEMENTATION AND EVALUATION

In this section we develop a proof-of-concept implementation of our redaction approach on JD Chain[5], evaluating the additional



(a) (b) (c) (d)

Figure 7: The above figure shows the overhead of our redactable approach through a proof-of-concept implementation on JD Chain. The figure in (a) shows the time overhead of issuing a new block with one redaction compared to an immutable chain; the figure in (b) shows the validation time overhead (in percentage) required to validate a chain for an increasing number of redactions compared to an immutable chain; the figure in (c) shows the validation time overhead (in percentage) required to validate an increasing chain with 10 redactions compared to an immutable chain; and finally the figure in (d) shows the time overhead of completing one redaction.

cost over the underlying immutable blockchain protocol. Specifically, we adopt the pairing-based multi-signature scheme in [10], and the general VRF scheme in [14] built from the unique signature which is instantiated with the unique BLS signature [11]. In the implementation we choose the security parameters for VRF and multi-signature to satisfy the 128-bit security level. The programme runs on a Lenovo Think-Station P318 computer with Ubuntu 16.04.10 (64bits) system, equipped with a 3.60 GHz Intel Core i7-7700 CPU with 8 cores and 32GB memory. Additionally, to evaluate the performance, we set $h = 0.75$, which means the adversary would control at most 25% of the stakes of the system, then the corresponding expected committee size is $T = 1000$ according to Figure 4.

Overhead of issuing new blocks with one redaction. In the first experiment, to evaluate the time overhead of the block issue process brought by one redaction, we generate the redactable blockchain and immutable blockchain with block size ranging from 500 up to 2000 transactions. The results in Figure 7a show that independent of the block size (or the number of transactions), there is an overhead of about 2 seconds on the block issue, including the overhead of checking voting right, voting for a redaction, and collecting votes. Intuitively, it is a non-negligible cost for block issue, however, it is acceptable in practice. On one hand, compared to the interval between two blocks (e.g., it takes about 60 seconds to produce a new block in JD Chain, and 10 minutes in Bitcoin), the issue of new block is not affected by 2 seconds; on the other hand, the redaction operation just occurs in particular and emergent cases under strict constraints rather than a frequent event, and thus the total overhead is minimal.

Overhead of validating a chain by the number of redactions. In the second experiment, we intend to evaluate the time overhead (in percentage) required to validate a redactable chain with respect to the number of redactions compared to an immutable chain. We generate the redactable chain consisting of 5000 blocks and each block contains 1000 transactions. The results in Figure 7b show that the overhead tends to be linear in the number of redactions, where the overhead mainly contains the validation time of the corresponding proofs for redactions.

Overhead of validating a chain by the chain size. In the third experiment, we intend to evaluate the time overhead (in percentage) required to validate a redactable chain with constant number of redactions and increasing chain size compared to an immutable chain. We set 10 redactions and each block contains 1000 transactions. The results in Figure 7c show that the overhead tends to be smaller with the increasing chain size, since the vote for one redaction can be validated by any user within just one block independent of subsequent blocks.

Overhead of completing one redaction by the chain size. In the last experiment, we intend to evaluate the time overhead required for one user to complete one redaction with increasing chain size, where each block contains 1000 transactions. The results in Figure 7d show that independent of the chain size, there is a nearly constant overhead of about 14 minutes to complete one redaction, including the time cost from the proposal to the final confirmation of the redaction. However, in [19], the voting period is required to be about 1024 consecutive blocks, which means about 17 hours to complete one redaction in JD Chain and about 7 days in Bitcoin. Therefore, our construction achieves significant efficiency improvement in fast confirmation.

Storage overhead compared to immutable blockchain. Compared to the immutable blockchain, for each block of our scheme, we store both of the initial and updated state of the block data, and thus one additional hash storage is needed. In addition, if one leader collects enough votes (i.e., $\frac{2}{3} \cdot T$) for an honest edit request in a slot, he/she would add the data $(msig, PROOF)$ to the new block, and the incremental storage of this block is at most $|msig| + |PROOF| = |msig| + \frac{2}{3} \cdot T(|H| + |\pi|)$, while the size of other blocks remains unchanged. According to the experiment results, the incremental storage is about 53.1 KB. Note that unless the leader handles more than one edit requests (e.g., l requests) in one slot, where the needed storage tends to be at most linear in l, the storage for several edits would be amortized among multiple blocks. Moreover, note that each VRF output from the stakeholder may represent several votes, which is determined by its stake weight, and thus the incremental storage cost may be much less than the above results.

Network delays. Recall that in our scheme, we set two time-out parameters, one for waiting time τ_t of the leader, and the other for the period t_p of validity of one edit request, to model various network environments.

The edit request would be invalid after a period of t_p from the beginning of being proposed, which may be due to the fact that the edit is adversarial and disapproved by honest users or the network environment is terrible and enough votes cannot be received. As a result, t_p should be set according to specific network environments. Specifically, t_p can be set to be a relatively small value in good environment with low latency, while for long-delay networks, it should be set appropriately larger to guarantee enough votes to a great extent.

The time window τ_t is set to guarantee the normal issue of new blocks. If the waiting time of the leader reaches τ_t, however received votes are not enough, then the leader would issue the new block as usual, leaving the edit request to next slot with double waiting time. Note that if the network environment is well enough, for example in full synchronous environment, then τ_t can be set to be a small value and the edit request can be approved within just a few slots (even only one slot). While in a relatively bad environment, it may cost more slots for one edit request to be approved until the request is invalid and revoked after a period of t_p.

In general, both t_p and τ_t are set based on the specific network environment and protocol instance. The system can be run normally under the cooperation of t_p and τ_t. Specifically, τ_t is initially set to be a small value and increased exponentially to ensure an honest edit request would be approved eventually even in the bad environment, while t_p restricts the maximum waiting time to guarantee the release of new blocks unaffected.

REFERENCES

[1] Akasha. https://akasha.world.
[2] Ethereum project. https://www.ethereum.org/.
[3] The EU general data protection regulation. https://gdpr-info.eu/.
[4] The illinois blockchain initiative. https://illinoisblockchain.tech.
[5] JD Chain. https://ledger.jd.com/.
[6] Steem. https://steem.
[7] Giuseppe Ateniese, Michael T Chiaramonte, David Treat, Bernardo Magri, and Daniele Venturi. 2018. Rewritable blockchain. uS Patent 9,967,096.
[8] 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.
[9] Christian Badertscher, Peter Gazi, Aggelos Kiayias, and Zikas Vassilis Russell, Alexander. 2018. Ouroboros Genesis: composable proof-of-stake blockchains with dynamic availability. In *Proceedings of ACM conference on Computer and communications security*. ACM, 913–930.
[10] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *ASIACRYPT 2018*, Vol. 11273. Springer, 435–464.
[11] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. *ASIACRYPT* (2001), 514–532.
[12] 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.
[13] CBinsights. 2018. Banking is only the beginning: 50 big industries blockchain could transform. https://www.cbinsights.com/research/ industries-disrupted-blockchain/.
[14] Jing Chen and Silvio Micali. 2017. Algorand. In *arXiv:1607.01341v9*.
[15] The NXT Community. 2014. NXT whitepaper. https://bravenewcoin.com/assets/Whitepapers/NxtWhitepaper-v122-rev4.pdf.
[16] Phil Daian, Rafael Pass, and Elaine Shi. 2019. Snow White: robustly reconfigurable consensus and applications to provably secure proof of stake. In *Proceedings of FC 2019*. Springer, 23–41.
[17] 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.
[18] 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*.
[19] Dominic Deuber, Bernardo Magriy, Sri Aravinda, and Thyagarajan Krishnan. 2019. Redactable blockchain in the permissionless setting. In *IEEE Symposium on Security and Privacy 2019*.
[20] 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.
[21] The Economist. 2017. Governments may be big backers of the blockchain. https://goo.gl/uEjckp.
[22] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert van Renesse. 2016. Bitcoin-NG: a scalable blockchain protocol. In *Proceedings of the 13th Symposium on Networked Systems Design and Implementation*. 45–59.
[23] Accenture files patent for editable blockchain. 2016. *Business Insider Deutschland*. https://tinyurl.com/yblq9zdp.
[24] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. 9057 (2015), 281–310.
[25] 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.
[26] 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.
[27] 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
[28] 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*. https://eprints.soton.ac.uk/422879/.
[29] Christoph Jentzsch. Decentralized autonomous organization to automate governance. https://download.slock.it/public/DAO/WhitePaper.pdf.
[30] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Proceedings of CRYPTO 2017*. Springer, 357–388.
[31] Sunny King and Scott Nadal. 2012. PPcoin: Peer-to-peer crypto-currency with proof-of-stake. https://peercoin.net/assets/paper/peercoin-paper.pdf.
[32] Jerin Mathew. 2015. Bitcoin: Blockchain could become 'safe haven' for hosting child sexual abuse images. http://www.dailydot.com/business/bitcoinchild-porn-transaction-code/.
[33] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. 2001. Accountable-subgroup multisignatures: Extended abstract. In *8th Conference on Computer and Communications Security*. ACM, 245–254.
[34] 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.
[35] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.
[36] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT 2017*, Vol. 10211. Springer, 643–673.
[37] Rafael Pass and Elaine Shi. 2017. The sleepy model of consensus. In *ASIACRYPT 2017*, Vol. 10625. Springer, 380–409.
[38] Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. 2017. μ chain: How to forget without hard forks. In *IACR Cryptology ePrint Archive, 2017/106*.
[39] 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.
[40] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *Proceedings of the 2015 Financial Cryptography and Data Security Conference*. Springer, 507–527.
[41] Giannis Tziakouris. 2018. Cryptocurrencies-A Forensic Challenge or Opportunity for Law Enforcement? An INTERPOL Perspective. *IEEE Security & Privacy* 16 (2018), 92–94.

A IMMUTABLE PROOF-OF-STAKE BLOCKCHAIN PROTOCOL

We now recall the immutable proof-of-stake 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 Proof-of-Stake Blockchain Protocol Γ'

On input initialization request init() from $\mathcal{Z}$:
  Let $(pk, sk) := \text{Sig.Gen}(1^\lambda)$
  Let $chain$ be genesis block B_0, $chain := B_0$
On receive $chain'$:
  Assert $|chain'| > |chain|$ and $chain'$ is valid;
  Let $chain := chain'$ and broadcast $chain$
For every slot sl':

- On receive input transactions(d') from $\mathcal{Z}$:
  - If eligible($\mathcal{P}$, s, sl')= 1, where $\mathcal{P}$ is the current node with the stake s:
    Let $B := (sl', st', d', \sigma')$, such that $st' = H(sl, G(st, d), \sigma)$ for $\text{Head}(chain) = (sl, st, d, \sigma)$ and $\sigma' = \text{Sig.Sign}(sk_{\mathcal{P}}; sl', G(st', d'))$;
    Let $chain := chain \| B$ and broadcast $chain$
- Output extract($chain$) to $\mathcal{Z}$, where extract outputs an ordered list of each block in $chain$

Figure 8. Immutable Proof-of-Stake Blockchain Protocol

B IDEAL IMMUTABLE PROOF-OF-STAKE BLOCKCHAIN PROTOCOL

We present the corresponding ideal functionality $\mathcal{F}'_{tree}$ (Figure 9) and the ideal immutable proof-of-stake protocol Π'_{ideal} (Figure 10) for Γ', by pruning the redaction operations from $\mathcal{F}_{tree}$ (c.f. Figure 5) and Π_{ideal} (c.f. Figure 6), respectively.

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

On init: tree := genesis, time(genesis) := 0
On receive leader($\mathcal{P}$, t) from $\mathcal{A}$ or internally:
  let s be the stake 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 $\Gamma[\mathcal{P}, t]$
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 |chain′| > |chain| and $\mathcal{F}'_{tree}$.verify(chain′) = 1
For every slot:
–receive input B from $\mathcal{Z}$
–if $\mathcal{F}'_{tree}$.extend(chain, i, B*) outputs "succ", then let chain[i] := B* and broadcast chain
–output chain to $\mathcal{Z}$

Figure 10. Ideal Proof-of-Stake Blockchain Protocol

C 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 $B_j = (sl_j, st_j, d_j, ib_j, \sigma_j)$, 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 $B_j^* = (sl_j, st_j, d_j^*, ib_j^*, \sigma_j)$. 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 4). 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.

procedure validateBlockExt(B)

Parse $B = (sl, st, d, ib, \sigma)$;
Parse $ib = ib^{(1)}||\ldots||ib^{(l)}$, where $ib^{(i)} \in \{0, 1\}^* \; \forall i \in [l]$;
Validate data d, **if** invalid **return** 0;
Validate the leader, **if** invalid **return** 0;
if the signature σ on $(sl, G(st, d), ib)$ or on $(sl, ib^{(1)}, ib^{(1)})$ is verified with vk
  then return 1;
else return 0.

Algorithm 4: The extended block validation algorithm

Valid Blockchains. To validate a chain, the users run the validateChainExt algorithm (Algorithm 5). The only difference between Algorithm 5 and 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 6). 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.

procedure validateChainExt(C)

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

Algorithm 5: The extended blockchain validation algorithm

procedure validateCandExt(C, B_j^*)

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

Algorithm 6: The extended candidate block validation algorithm

D REVIEWS FROM ACM CCS'19 AND IMPROVEMENT

We would like to thank the anonymous reviewers from CCS'19 for their very valuable comments. In this resubmission, our manuscript has been improved in the following ways.

(1) "*The paper talks about a policy P which seems to have been directly borrowed from [19] but no definition is given.*"
A formal definition of the redaction policy $\mathcal{RP}$ (c.f. Definition 4.1) has been added. Different from [19], we consider the number of votes embedded in a block.

(2) "*The collectVote procedure seems to wrong. why would you continue if the verification fails? What is the necessity of computing a multisig? The intuition paragraph just reads out the algorithms rather than give any real intuition for the running of the algorithms.*"
We have corrected the error in collectVote procedure. We have added the description on the necessity of computing multi-signature, that is, multi-signature can reduces the communication complexity and storage overhead for proof-of-stake blockchain.

(3) "*The paper goes on to use the same technique as in [19].*"
In our paper, to design a redactable proof-of-stake blockchain protocol, we utilize the idea in [19] to add a new entry. i.e. the initial state ib representing the initial and unedited state of block, to keep the link relation between two adjacent blocks no matter whether there exists any redaction. However, different from [19], in our work:
  (a) We introduce new methods for redaction process;
  (b) We adopt the simulation approach and define ideal functionality during the security analysis, which is more formal for the analysis of blockchain protocols;
  (c) By our approach, redaction can be confirmed faster, which is more attractive for the supervision of blockchain in practice.

(4) "*Thanks for this interesting work addressing the "right to be forgotten" issue with PoS-based blockchain. The protocol design, especially the utilization of staked "cryptographic sortition" is refreshing. But the claim that the proposed scheme works for all existing PoS blockchains is too hasty.* "
We have corrected the statement. The design of our protocol is compatible with current proof-of-stake blockchain such as Ouroboros[9, 17, 30], NXT[15], PPCoin[31], and Snow White[16], i.e., it can be implemented right now and requires only minimal changes to the current blockchain, block, or transaction structures. Actually, we tend to propose a solution for most current proof-of-stake blockchain protocols. This is achieved by adopting the general immutable proof-of-stake protocol (c.f. Appendix A) to capture the fundamental features without any specific restrictions. Based on this general abstract, our redactable protocol is also general. Of course, whether one proof-of-stake protocol is suitable for our approach may depend on the particular circumstance, e.g., whether the protocol accepts an overhead stemming our approach on the block size, the block issue and chain validation.

(5) "*The evaluation is insufficient. The best evaluation strategy is to evaluate a prototype system in a simulated PoS blockchain network.*"
We thank reviewers for the good advice. We have developed a proof-of-concept implementation of our redaction approach on JD Chain[5], evaluating the additional cost over the underlying immutable blockchain protocol. Specifically, we evaluate the overhead of issuing new blocks with one redaction, the overhead of validating a chain by the number of redactions, the overhead of validating a chain by the chain size, the overhead of completing one redaction by the chain size, the storage overhead compared to immutable blockchain, and network delays.

(6) "*The original proof is largely in a holistic fashion and not very rigorous.*"
We have adopted simulation approach and conducted a comprehensive security analysis instead of a "descriptive" analysis in the original paper. Specifically, first considers an idealized functionality $\mathcal{F}_{tree}$ that keeps track of all valid chains at any moment of time, 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.