Redactable Proof-of-Stake Blockchain with Fast Confirmation

Jing Xu  
Institute of Software, Chinese Academy of Sciences

Xinyu Li  
Institute of Software, Chinese Academy of Sciences

Lingyuan Yin  
Institute of Software, Chinese Academy of Sciences

Bingyong Guo  
Institute of Software, Chinese Academy of Sciences

Han Feng  
Institute of Software, Chinese Academy of Sciences

Zhenfeng Zhang  
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 cryptocurrencies. However, “Right to be Forgotten” has been imposed in new European Union’s General Data Protection Regulation, making legally impossible to use immutable blockchains. Moreover, 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 uses a novel mechanism based on verifiable random functions to randomly select voters on different slots in a private and non-interactive way, and also offers public verifiability for redactable chains. Compared to previous solutions in permissionless setting, our redaction operation can be performed quickly, even only within one block in synchronous network, which is desirable for redacting harmful or sensitive data. Moreover, our protocol is compatible with current proof-of-stake blockchains requiring only minimal changes. Furthermore, 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 [33]. 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, 33, 37], 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 [10, 17, 18, 29], 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 cryptocurrencies 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) [8] in May 2018, it is no longer legally possible to use 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 [27]. Moreover, an immutable ledger is not appropriate for some new applications [15] that are being envisaged for the blockchain such as government and public records [3, 7] and social media [1, 5]. 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 [30], leaked private keys [36] and materials that infringe on intellectual rights [26]. More worse, immutability of blockchains facilitates illicit activities of international criminal groups, and brings the numerous challenges for law enforcement agencies such as Interpol [38]. 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 [28], 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 can only be confirmed after at least one week, 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. [9] firstly proposed the notion of redacting a blockchain. They use a chameleon hash function [14] 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 [4][6]. Recently, in order to support fine-grained and controlled redaction of blockchain, Derler et al. [19] 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 solution[s][9][19] 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. [35] also presented a redactable blockchain, called $\mu$ chain. In $\mu$ 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, $\mu$ chain also faces scalability problem when reconstructing decryption keys by the multi-party computation protocol.

Recently, Deuber et al. [20] 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. In our scheme we assume that the fraction of stakes held by honest users is above threshold $h$ (a constant greater than $\frac{1}{2}$). 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 $C$, and only committee members (in the new slot $sl$ of $C$) can vote for $B'_j$; if votes are approved by the editing policy (e.g., voted by the majority), the leader of $sl$ 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.

- Whether a certain stakeholder has right to vote 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.
- 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.
- 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, i.e., it can be implemented right now and requires only minimal changes to the current blockchain, block, or transaction structures. Our redaction approach is general, and all the cases of synchronous, semi-synchronous, and asynchronous network are considered. 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. [23] and Pass et al. [34]. In order to accommodate the edit operation, we introduce an extended definition called redactable common prefix considering the affect of edited data. Essentially, redactable common prefix requires that if the property of the common prefix is violated, it must be the case that there exist edited blocks satisfying the editing policy $P$. Then we prove that our redactable proof-of-stake blockchain protocol satisfies redactable common prefix,
chain quality and chain growth. We also explore how various attacks considered in practice can be addressed in our protocol. Specifically, we discuss unapproved editing, denial of service, and consensus delays.

**Performance Evaluation.** We instantiate VRF primitive, and conduct experiments evaluating the overhead of adding our redaction mechanism to proof-of-stake blockchain at both 128-bit and 192-bit security levels. The results show that compared to immutable blockchain, the overhead incurred for different numbers of redactions in the chain is minimal. Moreover, all signatures of voting for a edited block are aggregated a multi-signature, which drastically reduces the communication complexity for proof-of-stake blockchain.

2 PRELIMINARIES

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

2.1 Verifiable Random Functions

The concept of verifiable random functions is introduced by Micali et al.[32]. Informally, it is a pseudo-random function that provides publicly verifiable proofs of its outputs’ correctness.

**Definition 2.1 (Verifiable Random Functions)**[21]. A function family $F_{\ell}(\cdot) : \{0, 1\}^{\ell} \rightarrow \{0, 1\}^l$ is a family of VRFs if there exist algorithms (Gen, VRF, VerifyVRF) such that Gen outputs a pair of keys $(pk, sk)$; 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 VerifyVRF$_{pk}(x, y, \pi)$ verifies that $y = F_{sk}(x)$ using the proof $\pi$, 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 $\text{VerifyVRF}_{pk}(x, y_1, \pi_1, y_2, \pi_2)$ unless $y_1 = y_2$.
- **Provenance:** if $(y, \pi) = \text{VRF}_{sk}(x)$, then $\text{VerifyVRF}_{pk}(x, y, \pi) = 1$.
- **Pseudorandomness:** for any probabilistic polynomial time algorithm $A = (A_g, A_f)$, which runs for a total of $s(y)$ steps when its first input is $1^\lambda$, and does not query the oracle on $x$,

$$\Pr\left[b' \left| \begin{array}{l}
(pk, sk) \leftarrow \text{Gen}(1^\lambda); \\
(x, st) \leftarrow A^{\text{VRF}(\cdot)}_{sk}(pk); \\
y = \text{VRF}_{sk}(x); y_1 \leftarrow \{0, 1\}^{|\pi_{sk}|}; \\
b \leftarrow \{0, 1\}; b' \leftarrow A^{\text{VRF}(\cdot)}_{pk}(y, st)
\end{array} \right| \leq \frac{1}{2} + \text{negl}(y)\right.$$  

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 $\text{SIG} = (\text{Sig.Gen, Sig.Sign, Sig.Verify})$ with message space $M(\lambda)$ consists of the standard algorithms: key generation $\text{Sig.Gen}(\{1^\lambda\}) \rightarrow (pk, sk)$, signing $\text{Sig.Sign}(sk; m) \rightarrow \sigma$, and verification $\text{Sig.Verify}(pk; m, \sigma) \rightarrow 1$ if for all $(pk, sk) \leftarrow \text{Sig.Gen}(\{1^\lambda\})$ and $m \in M(\lambda)$.

To define security [25], we consider the following game between an adversary $\mathcal{A}$ and a challenger.

1. Setup Phase. The challenger chooses $(pk, sk) \leftarrow \text{Sig.Gen}(\{1^\lambda\})$.
2. Signing Phase. The adversary $\mathcal{A}$ sends signature query $m_1 \in M(\mathcal{A})$ and receives $\sigma_1 = \text{Sig.Sign}(sk; m_1)$.
3. Forgery Phase. $\mathcal{A}$ outputs a message $m$ and its signature $\sigma$. If $m$ is not queried during the Signing Phase and $\text{Sig.Verify}(pk; m, \sigma) = 1$, the adversary wins.

**Definition 2.2 (EUF-CMA).** We say that a signature scheme $\text{SIG}$ is existentially unforgeable under adaptive chosen-message attacks (EUF-CMA), if for all adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that

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

2.3 Multi-Signature Scheme

A multi-signature scheme [12, 31] is a protocol that enables the $n$ signers to jointly generate a short signature $\text{msg}$ on $m$ so that $\text{msg}$ convinces a verifier that all $n$ parties signed $m$.

A multi-signature scheme is defined as algorithms $\text{Pg, Kg, Sig, Vf}$. The system parameters $\text{par} \leftarrow \text{Pg}$ are generated by a trusted party. Each signer generates a pair of key $(pk_i, sk_i) \leftarrow \text{Kg(par)}$, and signers can collectively sign a message $m$ by each running the interactive algorithm $\text{Sign(par, PK, sk, m)}$, where $\text{PK}$ 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 $\sigma$. Algorithm $\text{KAg}$ outputs a single aggregate public key $\text{apk}$ on inputs a set of public keys $\text{PK}$. A verifier check the validity of a signature $\sigma$ on message $m$ under an aggregate public key $\text{apk}$ by calling the algorithm $\text{Vf(par, apk, m, \sigma)}$ 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 $(pk_i, sk_i) \leftarrow \text{Kg(par)}$ for $i = 1, \ldots, n$, and for any message $m$, if all signers input $\text{Sign(par, sk, m)}$, then they will output a signature $\sigma$ such that $\text{Vf(par, KAg(par, \{pk_i\}^n_{i=1}, m, \sigma)} = 1$.

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

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

3
We recall basic definitions. We start with a brief description of a proof-of-stake blockchain abstraction \( \Gamma \), and then describe how to extend \( \Gamma \) to a redactable blockchain protocol \( \Gamma' \).

### 3.1 Proof-of-Stake Blockchain Protocol

We recall basic definitions [18] of proof-of-stake blockchain. There are \( n \) stakeholders \( U_1, \ldots, U_n \) and each stakeholder \( U_i \) possesses \( s_i \) stake and a verification/secret key pair \((vk_i, sk_i)\). Without loss of generality, we assume that the verification keys \( vk_1, \ldots, vk_n \) 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 \( B := (sl, st, d, B_\sigma, \sigma) \), where \( sl \in \{sl_1, \ldots, sl_p\} \) is the slot number, \( st \in \{0, 1\}^k \) is the hash of the previous block, \( d \in \{0, 1\}^* \) is the block data, \( B_\sigma \) is a block proof containing information that allows stakeholders to verify if a block is valid, and \( \sigma \) is a signature on \((sl, st, d, B_\sigma)\) computed under the secret key of slot leader generating the block.

A blockchain \( C \) 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 the list of stakeholders identified by their public-keys, their respective stakes \((vk_1, s_1), \ldots, (vk_n, s_n)\) and auxiliary information. The length of a chain \( \text{len}(C) = m \) is its number of blocks. The block \( B_m \) is the head of the chain, denoted \( \text{Head}(C) \).

The blockchain protocol, denoted by \( \Gamma \), has a set of global parameters and a public set of rules for validation, and provides the nodes with the following functionalities:

- \( \Gamma\).updateChain(C): returns a longer and valid chain \( C' \) by retrieving new valid blocks from the network (if it exists).
- \( \Gamma\).validateChain(C): returns 1 if the chain is valid according to a public set of rules and 0 otherwise.
- \( \Gamma\).validateBlock(B): returns 1 if the block is valid according to a public set of rules and 0 otherwise.
- \( \Gamma\).broadcast(x): broadcasts \( x \) to all the nodes of the system.

**Definition 3.1 (Properties of Blockchain).** A blockchain protocol should satisfy the following three properties.

- **Common Prefix.** The chains \( C_1 \) and \( C_2 \) possessed by two honest parties at the onset of the slots \( sl_1 < sl_2 \) are such that \( C_1|_{sl_2} \preceq C_2 \), where \( C_i|_{sl_2} \) denotes the chain obtained by removing the last \( k \) blocks from \( C_i \) and \( \preceq \) denotes the prefix relation.
- **Chain Quality.** Consider any portion of length at least \( k \) of the chain possessed by an honest party at the onset of a round; the ratio of blocks originating from the adversary is at most \( 1 - \mu \). We call \( \mu \) the chain quality coefficient.
- **Chain Growth.** Consider the chains \( C_1 \) and \( C_2 \) possessed by two honest parties at the onset of two slots \( sl_1, sl_2 \) with \( sl_2 \) at least \( s \) slots ahead of \( sl_1 \). Then it holds that \( \text{len}(C_2) = \text{len}(C_1) \geq \tau \cdot s \). We call \( \tau \) the speed coefficient.

### 3.2 Redactable Blockchain Protocol

We construct our redactable blockchain protocol \( \Gamma' \) by modifying and extending the aforementioned protocol \( \Gamma \). First, an editing policy is introduced to determine whether an edit to the blockchain should be approved or not. Specifically, an edited block \( B' \) whose editing proposal in the slot \( sl \) is said to satisfy the policy, i.e., \( P(C, B') = 1 \), if the number of votes on \( B' \) is at least \( 2/3 \cdot t \), where \( t \) is a parameter that determines the expected number of stake in committee for voting whose selection will be discussed in Section 3.4.

Next, in order to accommodate editable data, we extend the above block structure to be of the form \( B := (sl, st, d, ib, B_\sigma, \sigma) \). Specifically, if a chain \( C \) with \( \text{Head}(C) = (sl, st, d, ib, B_\sigma, \sigma) \) is updated to a new longer chain \( C' = C||B' \), the newly created block \( B' = (sl', st', d', ib', B'_\sigma, \sigma') \) sets \( st' = H(sl, G(st, d), ib) \) and \( ib' = G(st', d') \), where \( H \) and \( G \) are prescribed collision-resistant hash functions, \( \sigma' \) is a signature on \((sl', G(st', d'), ib', B'_\sigma)\) computed under the secret key of slot leader generating the block \( B' \).

Notice that in order to maintain the link relationships between edited block and its neighbouring blocks, 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 \( B = (sl, st, d, ib, B_\sigma, \sigma) \). Then, \( \text{validateBlock}(\Gamma) \) and \( \text{validateChain}(\Gamma) \) need to be modified accordingly. Roughly speaking, we need to ensure that for an edited block, its original state before editing still can be accessible for verification.

**Validating Block.** To validate a block, the \( \text{validateBlock} \) algorithm (Algorithm 1) takes as input a block and first checks the validity of data included in the block according to the system rules. It then checks the validity of the leader by \( B_\sigma \). Finally, it verifies the signature \( \sigma \) with the verification key \( vk \) of the leader. In particular, for an edited block, the signature \( \sigma \) is on the "old" state \((sl, st, d, ib, B_\sigma)\).

```plaintext
procedure validateBlock(B)
Parse B = (sl, st, d, ib, B_\sigma, \sigma);
Validate data d, if invalid return 0;
Validate B_\sigma including the verification key vk of leader,
if invalid return 0;
if the signature \sigma on (sl, G(st, d), ib, B_\sigma) or on (sl, lb, lb, B_\sigma)
is verified with vk, then return 1;
else return 0.
```

**Algorithm 1:** The block validation algorithm

**Validating Chain.** To validate a chain, the \( \text{validateChain} \) algorithm (Algorithm 2) takes as input a chain \( C \) and first validates it from the head of \( C \). For every block \( B_j \), it first checks the validity of block \( B_j \), and then checks the relationship to the previous block \( B_{j-1} \), which has two cases depending on whether \( B_{j-1} \) is a edited block. If \( B_{j-1} \) has been redacted (i.e., \( st_{j-1} \neq H(sl_{j-1}, G(st_{j-1}, d_{j-1}, ib_{j-1})) \)), its check additionally depends on whether the editing policy \( P \) of the chain has been satisfied.

Additionally, the protocol \( \Gamma' \) provides three new functionalities: \( \text{validCand}, \text{checkVote}, \) and \( \text{collectVote} \).

\(^1\)It is required to wait \( k \) blocks to confirm a transaction in blockchain protocol.
Validating candidate editing block. To validate a candidate editing block \( B_j \) for the \( j \)-th block of chain \( C \), the validateCand algorithm (Algorithm 3) takes as inputs \( B_j \) and \( C \), and 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., \( st_{j+1} = H(sl_j, ib_j, ib_j) \).

Checking voting right. The checkVote algorithm (Algorithm 4) checks a stakeholder \( U_i \) (with secret key \( sk_i \) and stake \( s_i \)) whether having right to vote or not. Inspired by the idea of Algorand [24], it uses VRFs to randomly select voters in a private and non-interactive way. Specifically, \( U_i \) computes \( \text{validateCand}(B, C) \) with his own secret key \( sk_i \), where the pseudo-random hash determines how many votes of \( U_i \) 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, \( U_i \) with stakes \( s_i \) owns \( s_i \) units, each unit is selected with probability \( p = \frac{q}{T} \), 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 the expected number of votes in the system, \( T \) is the expected number of stakes in committee for voting, \( \sum_{q=0}^{s_i} B(q; s_i, p) = 1 \) and \( C(s_i, q) = \frac{n!}{q!(s_i-q)!} \). To determine how many sub-users of \( s_i \) in \( U_i \) are selected, the algorithm divides the interval \( [0,1) \) into consecutive intervals of the form \( \left[ \frac{q}{T}, \frac{q+1}{T} \right) \) for \( q \in \{0,1, \ldots, s_i\} \). If \( \text{hash} \) falls in the interval \( \left[ \frac{q}{T}, \frac{q+1}{T} \right) \), it means \( q \) sub-users (i.e., \( q \) votes) of \( U_i \) are selected, where \( \text{hash} \) is the bit-length of the hash.

Collecting votes. The collectVote algorithm (Algorithm 5) collects and validates the votes. The collected votes are stored in \( mgs \) buffer. To validate a vote, it first verifies the signature on \( H(B_{j+1}) \) under the verification key of the voter, and then verifies a proof \( \pi \) to confirm the voting right of the voter, i.e., \( \text{VerifyVRF}(\text{hash}, \pi, \text{seed} || \text{sl}) \). If the voter \( U_i \) was chosen \( k \) times (i.e., \( \text{hash} \) falls in the interval \( j^k \)), the number of votes from \( U_i \) is \( k \) as well. As soon as the number of votes collected is more than \( \frac{2/3 \cdot T}{\text{hash}} \), the algorithm generates a multi-signature \( \text{msg} \) on all these vote signatures \( SIG \), aggregates corresponding proofs \( \text{PROOF} \), and returns them. If not enough votes are collected within the allocated \( \tau \) time window, then the algorithm returns 0.

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

Redactable proof-of-stake blockchain protocol $\Gamma'$ is described in Figure 1. In the chain $C = (B_1, \cdots, B_m)$, a block is edited by the following steps.

1. If a user wishes to propose an edit to block $B_j$ in the chain $C$, he first parses $B_j = (sl_j, st_j, d_j, ib_j, B_{n-j}, \sigma_j)$, replaces $d_j$ with the new data $d'_j$, and broadcasts the candidate block $B'_j = (sl_j, st_j, d'_j, ib_j, B_{n-j}, \sigma_j)$ 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 $U_i$ first validates it by using $\Gamma'$.validateCand$(C, B'_j)$ (Algorithm 3), and stores it in his own editing pool $E\mathcal{P}$ if $B'_j$ is a valid candidate editing block. In the pool $E\mathcal{P}$, each candidate editing block has a period of validity $\tau_p$.

3. At the beginning of each new slot $sl$, every stakeholder $U_i$ tries to extend their local chain by using $\Gamma'$.validateCand$(C)$ to retrieve new valid blocks from the network. For every candidate editing block $B'_j$ in its own editing pool $E\mathcal{P}$, $U_i$ first checks whether $B'_j$ has expired or not, and if it is, $U_i$ removes $B'_j$ from $E\mathcal{P}$. Then $U_i$ computes $P(C, B'_j)$ to check whether $B'_j$ should be adopted in the chain, and if it outputs 1, $U_i$ replaces $B_j$ in the chain with $B'_j$ and removes $B_j$ from $E\mathcal{P}$. Finally, for every remaining candidate editing block $B'_j$ in the $E\mathcal{P}$, $U_i$ with stake $s_j$ checks whether he has voting right for this block in current slot $sl$ by using $\Gamma'$.checkVote$(sl, s_k, s_l, seed, T, S)$ (Algorithm 4), where $seed$ is a nonce generated for the slot $sl$, $T$ is a threshold that determines the expected number of stakes in committee for voting, and $S = \Sigma_i s_i$ is all the stakes in the system. If it holds, $U_i$ broadcasts $(hash, \sigma)$ and the signature $sig$ on $H(B'_j)$ with his own secret key $sk_i$.

4. The leader of new slot $sl$ collects and validates the votes by using $\Gamma'$.collectVote$(msgs, sl, seed, T, S, \tau_v)$ (Algorithm 5), where $\tau_v$ is the allowed maximum time of collecting votes in one slot. If it holds and returns $(msg, PROOF)$, the leader adds them to the data $d'$ and proposes a new block $B'_j$, where $d'$ is new block data collected.

Redactable proof-of-stake blockchain protocol $\Gamma'$ offers public verifiability. Concretely, to validate a redactable chain, users first check each block exactly like in the underlying immutable blockchain protocol $\Gamma$. If a "broken" link between blocks is found, then users check whether the link still holds for the old state information. In the approving case, users verify whether the edited block satisfies the editing policy $\mathcal{P}$ by checking the following blocks. For example, in the chain $C = (B_1, \cdots, B_m)$, if $st_j \neq H(sl_{j-1}, Ci(sl_{j-1}, d_{j-1}, ib_{j-1}))$, the chain is valid only under the condition of $st_j = H(sl_{j-1}, ib_{j-1}, ib_{j-1})$ and $\mathcal{P}(C, B_{j-1}) = 1$.

3.4 The Number of Committee for Redaction voting

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 $s_j$ units, then $U_i$ is regarded as $s_j$ 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 $p_s$ of a sub-user to be selected is $\frac{T}{S}$. Then the probability that exactly $K$ sub-users are sampled is

$$\left(\begin{array}{c} S \\ K \end{array}\right) p_s^K (1-p_s)^{S-K} = \frac{S!}{K!(S-K)!} \left(1 - \frac{T}{S}\right)^{S-K}$$

$$= \frac{S \cdot \cdots \cdot (S-K+1)}{S^K} \frac{T^K}{K!} \left(1 - \frac{T}{S}\right)^{S-K}$$

If $K$ is fixed, we have

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

and

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

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

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

(1)

When we select the value of $T$, we want the number of honest committee members is more than $I_k \cdot T$, where $I_k \cdot T$ are some predetermined threshold. The condition is violated when the number of honest committee members is not more than $I_k \cdot T$. 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 $h (h > 2/3)$. Thus, the probability that the condition is
satisfying the condition, as a function of $2$ with a negligible probability, since
violated is given by the formula

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

$F$ is a parameter which marks a negligible probability that the condition fails, and our experience sets $F = 5 \times 10^{-9}$. 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 $10^5$, and then see if we can find a $l_s \in \left(\frac{5}{4}, 1\right]$ 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 2, plotting the expected committee size $T$ satisfying the condition, as a function of $h$, with a probability of violation of $5 \times 10^{-9}$. A similar approach to compute the threshold of committee size can be referred to [24].

In the implementation of our system, we assume the fraction of honest stakes is 0.75, so we select $T = 1000$ according to Figure 2. From Algorithm 5, a validate editing block is approved only after it obtains more than $\frac{1}{2} \cdot T$ votes.

We stress that the number of votes from malicious stakeholders cannot reach $2/3 \cdot T$ 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 $2/3 \cdot n$ with probability at least $1 - F$ according to the above discussion, while the malicious committee members can only reach $2/3 \cdot T$ unless $1/3 \cdot n > 2/3 \cdot T$ (i.e., $n > 2T$), which occurs with a negligible probability since $T$ is the expected value of the committee size following the binomial distribution. Similarly, when $n < T$, the malicious members can only obtain more than $2/3 \cdot T$ votes unless $\rho' \cdot n > 2/3 \cdot T$ (i.e., $\rho' > 2/3$), where $\rho'$ 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 $1/3$ too much. This result keeps consistent with that in Algorand [24].

4 SECURITY ANALYSIS

In this section, we analyze the security of redactable proof-of-stake blockchain protocol $\Gamma'$ as depicted in Figure 1.

Essentially, $\Gamma'$ behaves just like the underlying immutable proof-of-stake blockchain protocol $\Gamma$ if there is no edit in the chain, and otherwise each edit must be approved by the policy $P$. Therefore, we prove $\Gamma'$ preserves the same properties (or a variation of the property) of the underlying immutable PoS blockchain protocol $\Gamma$ under the editing policy $P$, that is, $\Gamma'$ satisfies the properties of redactable common prefix, chain quality and chain growth.

**Common Prefix.** We observe that redactable proof-of-stake protocol $\Gamma'$ inherently does not satisfy the original definition of common prefix due to the (possible) edit operation. In detail, consider the case where two chains $C_1$ and $C_2$ are held by two honest parties $P_1$ and $P_2$ at slot $sl_1$ and $sl_2$ respectively, such that $sl_1 < sl_2$. For a candidate block $B^*_j \in C_1$ 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 $C_1$ but may have taken effect in $C_2$. As a result, the original block $B_j$ remains unchanged in $C_1$ while it is replaced with the candidate $B^*_j$ in $C_2$. Therefore, $C_1^k \neq C_2$, 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 common prefix property. To address this issue, we introduce an extended definition called redactable common prefix, which is suitable for redactable blockchains and considers the affect of each edit. 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 $P$.

**Definition 4.1.** (Redactable Common Prefix). The chains $C_1$ and $C_2$ of length $l_1$ and $l_2$, respectively, possessed by two honest parties at the onset of the slots $sl_1 < sl_2$ satisfy one of the following:

1. $C_1^k \neq C_2$, or
2. for each $B^*_j \in C_2_{(l_2-l_1)+k}$ such that $B^*_j \notin C_1^k$, then it must be the case that $P(C_2, B^*_j) = \text{accept}$, for $j \in [l_1 - k]$, where $C_2_{(l_2-l_1)+k}$ denotes the chain obtained by removing the last $(l_2 - l_1) + k$ blocks from $C_2$, namely the first $l_1 - k$ blocks of $C_2$. $P$ denotes the editing policy, and $k$ denotes the common prefix parameter.

Now we prove the redactable PoS blockchain protocol $\Gamma'$ satisfies the redactable common prefix property.

**Theorem 4.2.** If the hash function $H$ is collision resistant, and the immutable blockchain protocol $\Gamma$ satisfies common prefix property, then $\Gamma'$ satisfies the redactable common prefix property.

**Proof.** Note that if there is no edit in the chain $C$, then $\Gamma'$ behaves exactly like the immutable blockchain protocol $\Gamma$, and thus the common prefix property (cf. Definition 3.1) can be preserved directly.
In case of an edit, we consider a new candidate block $B'_j$ for the original block $B_j$ in chain $C_j$, which is later edited by honest $P_2$. We can observe that the adversary cannot propose another candidate $B'_j \neq B'_j$ such that $H(B'_j) = H(B'_j)$, since this would break the collision resistance property of the hash function $H$. Therefore, if the honest $P_2$ eventually replaces $B_j$ with $B'_j$, then it must be the case that $P_2$ receives enough votes for $B'_j$ according to the protocol specification. This concludes the proof.

**Chain Quality.** The chain quality property restricts the ratio of adversarial blocks to a fraction $\mu$, where $\mu$ denotes the fraction of stakes controlled by the adversary. We prove that $\Gamma'$ satisfies the chain quality property as follows.

**Theorem 4.3.** If the hash function $H$ is collision resistant, the signature scheme $Sig$ is EUF-CMA secure, the multi-signature scheme $Msig$ is MEUF-CMA secure, and the immutable blockchain protocol $\Gamma$ satisfies the chain quality property with parameters $(k, \mu)$, then $\Gamma'$ satisfies the chain quality property with parameters $(k, \mu)$.

*Proof:* Note that if there is no edit in the chain, then $\Gamma'$ behaves exactly like the immutable blockchain protocol $\Gamma$, and thus the chain quality property (cf. Definition 3.1) can be preserved directly.

In case of an edit, the adversary $\mathcal{A}$ can propose an edit request to replace an honest block $B_j$ with a malicious block $B'_j$ (e.g., containing illegal or harmful data), and by this way, $\mathcal{A}$ can increase the proportion of adversarial blocks in the chain and finally break the chain quality property. We will show $\mathcal{A}$ can break the chain quality property only with a negligible probability, if the hash function $H$ is collision resistant, the signature signature scheme $Sig$ is EUF-CMA secure and the multi-signature scheme $Msig$ is MEUF-CMA secure.

**Case-I:** If $\mathcal{A}$ wants to edit an honest block $B_j$ into adversarial block $B'_j$, he will try to build and propose an “honest looking” candidate block $B''$ to replace $B_j$ such that $H(B'') = H(B'_j)$. The honest nodes could endorse the honest candidate block $B''$ during the voting process, however, $\mathcal{A}$ just maliciously edits the $B_j$ into $B'_j$ rather than the adopted $B''$. Note that, the edit by the adversary $\mathcal{A}$ is valid according to the protocol specification. However, $\mathcal{A}$ has only a negligible probability to generate such a candidate block $B''$ such that $H(B'') = H(B'_j)$, since this would break the collision resistance property of the hash function $H$.

**Case-II:** We will show that $\mathcal{A}$ cannot employ the ability of adaptively corrupt during the voting process to vote for his adversarial request. The adversarial edit request (e.g., editing $B_j$ into $B'_j$) can only be adopted when the number of votes reaches $2/3 - T$, while $\mathcal{A}$ himself has no enough votes. 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 $2/3 - T$ 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 $1/2^{hashlen}$.

**Case-III:** As described above, the adversarial edit request (e.g., edit $B_j$ into $B'_j$) can only be adopted when the number of votes exceeds $2/3 - T$, however the adversary has no enough votes.

During the voting for some honest candidate block in current slot, the adversary $\mathcal{A}$ can confirm that $U_i$ belongs to the voting committee by eavesdropping, and further obtain the valid $PROOF$ of the honest user $U_i$ from the channel. If $\mathcal{A}$ can forge the signature of $U_i$ to vote the adversarial edit request, he may obtain enough votes by adding the vote of $U_i$. Then he can propose an adversarial edit block by broadcasting $2/3 - T$ votes, and finally this adversarial edit would be adopted with enough votes. However, the advantage of $\mathcal{A}$ is negligible due to the EUF-CMA security property of the underlying signature scheme $Sig$. If $\mathcal{A}$ succeeds by this way, then we can construct another algorithm $\mathcal{B}$ to break the EUF-CMA security of $Sig$. Generally this is achieved by a reduction, that is, $\mathcal{B}$ simulates for $\mathcal{A}$ the protocol running just as the protocol specification. For any signature to generate for user $U_i$ in honest sessions, $\mathcal{B}$ calls its signing oracle in its own EUF-CMA experiment. Eventually, if $\mathcal{A}$ outputs a valid signature $\sigma$ from $U_i$ and $\sigma$ has never been previously output by the signing oracle, $\sigma$ can be used as a forgery and EUF-CMA security of $Sig$ is broken.

**Case-IV:** We consider the case where one of users controlled by the adversary $\mathcal{A}$ is selected as the leader in the current slot. In order to make an adversarial candidate block adopted, $\mathcal{A}$ needs to produce a valid multi-signature from $2/3 - T$ users on the adversarial edit request message, however, among the $2/3 - T$ users at least one is honest according to the protocol specification.

To achieve this goal, $\mathcal{A}$ first collects the votes and the corresponding proof for some honest edit, then he can confirm that some honest users such as $U_i$ belong to the voters committee and have the voting right in current slot. Then $\mathcal{A}$ tries to produce a multi-signature $msig$ on the adversarial edit request from $2/3 - T$ users including honest users such as $U_i$ and adversarial users, and adds the $msig$ as well as the corresponding proof $PROOF$ into the new block. It is easy to see that both $msig$ and $PROOF$ are valid and thus the adversarial edit request would be adopted. However, the probability of $\mathcal{A}$ to produce a multi-signature $msig$ from $2/3 - T$ users one of which is at least honest is negligible, due to the MEUF-CMA security of the underlying multi-signature scheme $Msig$. Specifically, if $\mathcal{A}$ successfully produces such a valid $msig$, then we can construct another algorithm $\mathcal{B}$ to break the MEUF-CMA security of $Msig$. During the reduction, $\mathcal{B}$ simulates the protocol running for $\mathcal{A}$ just as the protocol specification. For any signature to generate for user $U_i$ in honest sessions, $\mathcal{B}$ calls the signing oracle of $U_i$ as specified in the MEUF-CMA security experiment. Eventually, if $\mathcal{A}$ outputs a valid multi-signature $msig$ on some message $m$ and $m$ has never been queried to the signing oracle of $U_i$, $msig$ can be used as a forgery and MEUF-CMA security of $Msig$ is broken. This concludes the proof.

**Chain Growth.** This property requires the chain grows proportionally with the number of rounds of the protocol.

**Theorem 4.4.** If $\Gamma$ satisfies the chain growth property with parameters $(r, s)$, then $\Gamma'$ also satisfies the chain growth property with parameters $(r, s)$ under the editing policy $P$.
Proof. Note that any edit operation in $\Gamma'$ would not reduce the length of the chain since it is not possible to remove any blocks from the chain according to the protocol specification. Moreover, the new block issue process in current slot is not influenced by votes for any edit request, since the leader would always issue new blocks in current slot no matter whether he/she has received enough votes within pre-defined time window. Therefore, we conclude $\Gamma'$ preserve the chain growth property of $\Gamma$. □

5 IMPLEMENTATION AND EVALUATION

We make an evaluation of our redactable blockchain protocol, in terms of additional cost over the underlying immutable blockchain protocol. Specifically, we implement the new added cryptographic primitives, including VRF scheme and multi-signature scheme, to evaluate the additional storage cost for a block and the additional computation time over the system.

We adopt the pairing-based multi-signature scheme in [12], while for VRF, we adopt the general scheme [16] built from the unique signature which is instantiated in this paper with the unique BLS signature [13]. The corresponding implementations are written in C using version 3 of AMCL and compiled using gcc 5.4.0, and 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. Particularly, the AMCL library recommends two types of BLS curves (i.e., BLS12 and BLS24) to support bilinear pairings, and the curves have the form $y^2 = x^3 + b$ defined over a finite field $\mathbb{F}_q$, with $b = 15$ and $|q| = 383$ for BLS12, while $b = 19$ and $|q| = 479$ for BLS24, where $q$ is a prime. According to the analysis [11], BLS12 and BLS24 curves can provide 128-bit and 192-bit security levels, respectively.

Table 1 and Table 2 summarize the experiment cost/size of each basic operation and each element over recommended groups at different security levels, where we use $t_{pr}$, $t_{vrf}$, $t_s$, $t_v$, and $t_a$ to denote the time for VRF computation, VRF validation, generating a signature, verifying a signature and aggregating two signatures, respectively. We also denote $|H|$, $|\sigma|$ and $|msig|$ as the bit-length of hash function, the output proof of VRF and an aggregated signature.

To evaluate the performance, we set $h = 0.75$, which means the adversary would control 25% of the stakes of the system, then the corresponding expected committee size is $T = 1000$ according to Figure 2.

<table>
<thead>
<tr>
<th>t_{pr}</th>
<th>t_{vrf}</th>
<th>t_s</th>
<th>t_v</th>
<th>t_a</th>
</tr>
</thead>
<tbody>
<tr>
<td>128-bit</td>
<td>0.46</td>
<td>1.52</td>
<td>0.49</td>
<td>1.52</td>
</tr>
<tr>
<td>192-bit</td>
<td>0.68</td>
<td>5.29</td>
<td>0.85</td>
<td>5.29</td>
</tr>
</tbody>
</table>

Table 2: Experimental size of each element (bits)

| | | | | |
|--------|--------|-----|-----|
| 128-bit | 256    | 381 | 381 |
| 192-bit | 384    | 478 | 478 |

Storage overhead for one block. Compared to the immutable blockchain, in 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., $2/3 \cdot T$) for an honest edit request in a slot, then he/she would add the data ($msig$, PROOF) to the new block, and the incremental storage of this block is at most $|msig| + |PROOF| = |msig| + 2/3 \cdot T(|H| + |\sigma|)$, while the size of other blocks remains unchanged. According to the experiment results, the incremental storage is about 53.1 KB and 71.9 KB for 128-bit and 192-bit security, respectively. Note that unless the 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 denote several votes, which is determined by its stake weight, as a result, there are no more than $2/3 \cdot T$ VRF outputs in the new slot, and the incremental storage cost may be much less than the above results.

Computation overhead for the system. Upon receiving an honest request, each user would check whether having right to vote or not in the current slot by running the checkVote algorithm in a parallel way, which leads to an additional time cost $t_{pr}$ for VRF evaluation and $t_s$ for signing to vote. For the leader that receives $2/3 \cdot T$ votes, he/she would check the vote right of each user, verify at most $2/3 \cdot T$ signatures, and aggregate the individual signatures into a single one $msig$, which costs about $2/3 \cdot T \cdot t_{pr}$, $2/3 \cdot T \cdot t_v$, and $2/3 \cdot T \cdot t_a$, respectively. Thus, the incremental computation cost $t$ for one edit is about 2.03s and 7.06s for 128-bit and 192-bit security respectively. As mentioned above, each signature and each VRF proof from the stakeholder may denote several votes, therefore, the leader would verify at most $2/3 \cdot T$ signatures and VRF outputs, which means in practice the incremental time cost may be much less than the above results. In addition, the incremental computation cost only exists in slots where edit request is being managed while other slots remain unaffected.

Network delays. Recall that in our scheme, we set two time-out parameters, one for waiting time $\tau_r$ 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 $\tau_r$ is set to guarantee the normal issue of new blocks. If the waiting time of the leader reaches $\tau_r$, 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 $\tau_r$ 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 $\tau_j$ are set based on the specific network environment and protocol instance. The system can be run normally under the cooperation of $t_p$ and $\tau_j$. Specifically, $\tau_j$ 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.

### 6 DISCUSSION AND MULTIPLE REDACTIONS

In this section, we discuss some possible attacks in practice and demonstrate how these attacks can be avoided in our protocol. We also extend our protocol to support multiple redactions per block.

#### 6.1 Discussion on Attacks in Practice

**Unapproved editing.** If leaders of some slots are malicious, they may edit the blockchain with some invalid candidate blocks that do not satisfy the editing policy $\mathcal{P}$. For example, malicious leaders edit the blockchain with candidate blocks that have gathered insufficient votes. However, other honest users will verify the blockchain and check whether every edit is approved or not. So honest users will reject the blockchain with unapproved edits.

**Denial of service.** Malicious users may launch denial of service attack by flooding the network with many edit requests. However, users need to spend a transaction fee for each edit request similar to other standard transactions. So we can immune to suck attack by making a higher transaction fee for each edit request.

**Consensus delays.** If two different miners maintain chains with a different set of redacted blocks, have been approved by the policy, which may result in consensus delays. However, this would violate the readable common prefix property in our protocol.

**Increasing the time window indefinitely.** In our system, a candidate editing block which does not obtain enough votes will be voted again in the next slot, but the time window increase exponentially with slot. Malicious users can make the time window very long by proposing some invalid candidate editing blocks that can not obtain sufficient votes, which slows down the efficiency of the system. However, each candidate editing block has a period of validity in our system. Therefore, when the time window is more than the period of validity, we reset the time window as $t$ to prevent it from increasing indefinitely.

### 6.2 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 $B_j = (s_j, st_j, d_j, ib^j, B_{\pi,j}, \sigma_j)$. Note that, if $B_j$ has never been redacted before, then $ib^j = G(st_j, d_j)$ and thus $ib^j = G(st_j, d_j)$.

**Validating block.** To validate a block, the users run the validateBlockExt algorithm (Algorithm 6). 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.

**Validating chain.** To validate a chain, the users run the validateChainExt algorithm (Algorithm 7). The only difference between Algorithm 7 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.
The extended candidate block validation algorithm

Algorithm 8: The extended candidate block validation algorithm

REFERENCES

[38] Rewritable blockchain. May 8 2018, us Patent 9,976,096.