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
yinlingyuan@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 impossible to use 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 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 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) [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 solutions[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 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. [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 $\frac{2}{3}$). 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 $\mathcal{P}$. 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 $negl(\cdot) : \mathbb{N} \to (0,1)$ 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.[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 $F_{(\cdot)}(\cdot) : \{0,1\}^{l} \to \{0,1\}^{l_{VRF}}$ 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,

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

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 [25], 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

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

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 *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 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 $i = 1, \ldots, n$, and for any message m, if all signers input $\mathsf{Sign}(par, sk_i, m)$, then they will output a signature σ such that $\mathsf{Vf}\left(par, \mathsf{KAg}\left(par, \{pk_i\}_{i=1}^{n}\right), m, \sigma\right) = 1$.

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

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

3 REDACTING THE BLOCKCHAIN

In this section we present a generic construction that converts any existing proof-of-stake blockchain into redactable blockchain. We start with a brief description of a proof-of-stake blockchain abstraction Γ, and then describe how to extend Γ to a redactable blockchain protocol Γ'.

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 $B := (sl, st, d, B_\pi, \sigma)$, 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, B_π is a block proof containing information that allows stakeholders to verify if a block is valid, and σ is a signature on (sl, st, d, B_π) 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 $len(C) = m$ is its number of blocks. The block B_m is the head of the chain, denoted $\mathsf{Head}(C)$.

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

- $\Gamma.\mathsf{updateChain}(C)$: returns a longer and valid chain C' by retrieving new valid blocks from the network (if it exists).
- $\Gamma.\mathsf{validateChain}(C)$: returns 1 if the chain is valid according to a public set of rules and 0 otherwise.
- $\Gamma.\mathsf{validateBlock}(B)$: returns 1 if the block is valid according to a public set of rules and 0 otherwise.
- $\Gamma.\mathsf{broadcast}(x)$: 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 $C_1^{\lceil k} \preceq C_2$, where $C_1^{\lceil k}$ denotes the chain obtained by removing the last k blocks from C_1 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 μ 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 $len(C_2) - len(C_1) \geq \tau \cdot s$. We call τ the speed coefficient.

3.2 Redactable Blockchain Protocol

We construct our redactable blockchain protocol Γ' by modifying and extending the aforementioned protocol Γ. 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 proposed in the slot sl is said to satisfy the policy, i.e., $\mathcal{P}(C, B^*) = 1$, if the number of votes on B^* is at least $2/3 \cdot T$, where votes are embedded in a block B_r, $B_r \in C^{\lceil k}$, and T is a parameter that determines the expected number of stake in committee for voting whose selection will be discussed in Section 3.4[1].

Next, in order to accommodate editable data, we extend the above block structure to be of the form $B := (sl, st, d, ib, B_\pi, \sigma)$. Specifically, if a chain C with $\mathsf{Head}(C) = (sl, st, d, ib, B_\pi, \sigma)$ is updated to a new longer chain $C' = C \| B'$, the newly created block $B' = (sl', st', d', ib', B'_\pi, \sigma')$ sets $st' = H(sl, G(st, d), ib)$ and $ib' = G(st', d')$, where H and G are prescribed collision-resistent hash functions, σ' is a signature on $(sl', G(st', d'), ib', B'_\pi)$ 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 $B = (sl, st, d, ib, B_\pi, \sigma)$. Then, $\mathsf{validateBlock}$ (Algorithm 1) and $\mathsf{validateChain}$ (Algorithm 2) 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 $\mathsf{validateBlock}$ 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_π. Finally, it verifies the signature σ with the verification key vk of the leader. In particular, for an edited block, the signature σ is on the "old" state (sl, ib, ib, B_π).

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

Algorithm 1: The block validation algorithm

Validating Chain. To validate a chain, the $\mathsf{validateChain}$ 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., $st_j \neq H(sl_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1})$), its check additionally depends on whether the editing policy $\mathcal{P}$ of the chain has been satisfied.

Additionally, the protocol Γ' provides three new functionalities $\mathsf{validCand}$,$\mathsf{checkVote}$, and $\mathsf{collectVote}$.

[1]It is required to wait k blocks to confirm a transaction in blockchain protocol.

procedure validateChain(C)
Parse $C = (B_1, \cdots, B_m)$;
$j = m$;
if $j = 1$ **then return** Γ'.validateBlock(B_1);
while $j \geq 2$ **do**
$B_j = (sl_j, st_j, d_j, ib_j, B_{\pi,j}, \sigma_j)$;
if Γ'.validateBlock(B_j) = 0 **then return** 0;
if $st_j = H(sl_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1})$ **then**
$j = j - 1$;
else if $st_j = H(sl_{j-1}, ib_{j-1}, ib_{j-1})$ and $\mathcal{P}(C, B_{j-1}) = 1$
then $j = j - 1$;
else return 0;
return 1.

Algorithm 2: The chain validation algorithm

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., $st_{j+1} = H(sl_j, ib_j, ib_j)$.

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

Algorithm 3: The candidate block validation algorithm

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 $(hash, \pi) \leftarrow VRF_{sk_i}(seed\|sl)$ with his own secret key sk_i, where the pseudo-random *hash* determines how many votes of 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 $p = \frac{T}{S}$, 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, $\Sigma_{q=0}^{s_i} B(q; s_i, p) = 1$ and $C(s_i, q) = \frac{s_i!}{q!(s_i-q)!}$. 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 $I^c = [\Sigma_{q=0}^{c} B(q; s_i, p), \Sigma_{q=0}^{c+1} B(q; s_i, p))$ for $c \in \{0, 1, \cdots, s_i\}$. If $\frac{hash}{2^{hashlen}}$ falls in the interval I^c, it means that c sub-users (i.e., c votes) of U_i are selected, where *hashlen* is the bit-length of *hash*.

Collecting votes. The collectVote algorithm (Algorithm 5) 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 verification key of the voter, and then verifies a proof π to confirm the voting right of the voter, i.e., $VerifyVRF_{vk}(hash, \pi, seed\|sl)$[2]. If the voter U_i was chosen k times (i.e., $\frac{hash}{2^{hashlen}}$ falls in the interval I^k), the number of votes from U_i is k as well. As soon as the number of votes collected is more than $2/3 \cdot T$, the algorithm generates a multi-signature *msig* on all these vote signatures *SIG*, aggregates corresponding proofs *PROOF*, and returns them. If not enough votes are collected within the allocated τ_t time window, then the algorithm returns 0.

procedure 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.

Algorithm 4: Checking voting right

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 sets $\tau_t = t$, 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 $\tau_t = 2t$. 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.

procedure collectVote($msgs$, sl, $seed$, T, S, τ_t)
$start \leftarrow Time()$;
$votes \leftarrow 0$;
$SIG \leftarrow \{\}$;
$PROOF \leftarrow \{\}$;
For every $m \leftarrow msgs.next()$
if $Time() > start + \tau_t$ **then return** 0;
else
$(hash, \pi, sig) \leftarrow m$;
if the signature sig on $H(B_j^*)$ is not verified
then continue;
if $VerifyVRF_{vk}(hash, \pi, seed\|sl) = 0$
then continue;
$p \leftarrow \frac{T}{S}$;
$c \leftarrow 0$;
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**
$c \leftarrow c + 1$;
$votes = votes + c$;
$SIG = SIG \cup \{sig\}$;
$PROOF = PROOF \cup \{(hash, \pi)\}$;
if $votes > 2/3 \cdot T$
then compute multi-signature $msig$ on SIG
and **return** $(msig, PROOF)$.

Algorithm 5: Collecting votes

[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.

3.3 Protocol Description

Redactable proof-of-stake blockchain protocol Γ' 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 $B_j = (sl_j, st_j, d_j, ib_j, B_{\pi,j}, \sigma_j)$, replaces d_j with the new data d_j^*, and broadcasts the candidate block $B_j^* = (sl_j, st_j, d_j^*, ib_j, B_{\pi,j}, \sigma_j)$ 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 Γ'.validateCand(C, B_j^*) (Algorithm 3), and stores it in his own editing pool $\mathcal{EP}$ if B_j^* is a valid candidate editing block. In the pool $\mathcal{EP}$, each candidate editing block has a period of validity t_p.

(3) At the beginning of each new slot sl, every stakeholder U_i tries to extend their local chain by using Γ'.validateCand(C) to retrieve new valid blocks from the network. For every candidate editing block B_j^* in his own editing pool $\mathcal{EP}$, U_i first checks whether B_j^* has expired or not, and if it is, U_i removes B_j^* from $\mathcal{EP}$. Then U_i computes $\mathcal{P}(C, B_j^*)$ 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 $\mathcal{EP}$. Finally, for every remaining candidate editing block B_j^* in the $\mathcal{EP}$, U_i with stake s_i checks whether he has voting right for this block in current slot sl by using Γ'.checkVote$(sl, sk_i, s_i, seed, T, S)$ (Algorithm 4), where $seed$ is a nonce generated for the slot sl[3], 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 $(hash, \pi)$ 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 Γ'.collectVote$(msgs, sl, seed, T, S, \tau_t)$ (Algorithm 5), where τ_t is the allowed maximum time of collecting votes in one slot. If it holds and returns $(msig, PROOF)$, the leader adds them to the data d' and proposes a new block B', where d' is new block data collected.

Redactable proof-of-stake blockchain protocol Γ' offers public verifiability. Concretely, to validate a redactable chain, users first check each block exactly like in the underlying immutable blockchain protocol Γ. 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 $st_j \neq H(sl_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1})$, the chain is valid only under the condition of $st_j = H(sl_{j-1}, ib_{j-1}, ib_{j-1})$ and $\mathcal{P}(C, B_{j-1}) = 1$.

The protocol Γ' is run by stakeholders over a sequence of slots sl, and is parameterized by editing policy $\mathcal{P}$, corrupted stakes ratio ρ, and expected number of stakes in voters committee T, where $\rho = 1 - h < 1/3$.

Initialization. Set the chain C be genesis block B_0.
Chain update. At the beginning of a new slot sl, the nodes try to update their local chain by calling $C \leftarrow \Gamma'$.updateChain.
Editing pool update. Collect all candidate editing blocks B_j^* from the network, and add B_j^* to the editing pool $\mathcal{EP}$ iff Γ'.validateCand$(C, B_j^*) = 1$; otherwise discard B_j^*. Remove every candidate editing block in $\mathcal{EP}$ which has expired.
Editing the chain. For every candidate block B_j^* in $\mathcal{EP}$, if $\mathcal{P}(C, B_j^*) = 1$, the block B_j in C is replaced by B_j^*.
Creating a new block. The leader in a new slot sl performs the following steps:

- Collect all the transaction data d' from the network.
- Collect the votes for candidate editing block $H(B_j^*)$(if exists) by calling Γ'.collectVote, and add votes and proofs to d' provided that the number of votes is more than $2/3 \cdot T$.
- Create a new block $B = (sl', st', d', ib', B'_\pi, \sigma')$, such that $st' = H(sl, G(st, d), ib)$ for Head$(C) = (sl, st, d, ib, B_\pi, \sigma)$.
- Extend its local chain $C \leftarrow C \| B$ and then broadcast C to the network.

Proposing an edit. The stakeholder creates a candidate block B_j^* using new data d^*, and broadcasts it to the network.
Voting for candidate editing blocks. For every candidate block B_j^* in $\mathcal{EP}$, the stakeholder checks his own voting right by calling Γ'.checkVote, then votes for B_j^* and broadcasts voting information to the network.

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

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_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

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

If K is fixed, we have

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

and

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

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

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

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 ł$_s \cdot T$. 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 ($h > 2/3$). Thus, the probability that the condition is

[3] In proof-of-stake blockchain protocol, in order to guarantee the adversary cannot control the selection of the leader, random seed needs to be introduced by different ways.

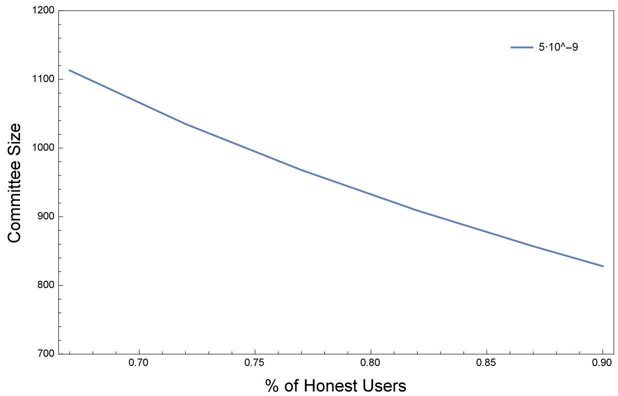

Figure 2. The x-axis specifies h, the stakes fraction of honest users. The committee size, T, is sufficient to limit the probability of violating safety to 5×10^{-9}.

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 $F = 5 \times 10^{-9}$. 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 10^4, 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 2, 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 [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 $\frac{2}{3} \cdot T$ 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 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 $1/3$ 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 Γ' as depicted in Figure. 1.

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

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 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^* 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, $C_1^{\lceil k} \npreceq C_2$, 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 $\mathcal{P}$.

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) $C_1^{\lceil k} \preceq C_2$, or
(2) for each $B_j^* \in C_2^{\lceil (l_2 - l_1)+k}$ such that $B_j^* \notin C_1^{\lceil k}$, then it must be the case that $\mathcal{P}(C_2, B_j^*) = $ accept, for $j \in [l_1 - k]$, where $C_2^{\lceil (l_2 - l_1)+k}$ 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, $\mathcal{P}$ denotes the editing policy, and k denotes the common prefix parameter.

Now we prove the redactable PoS blockchain protocol Γ' satisfies the redactable common prefix property.

THEOREM 4.2. *If the hash function H is collision resistant, and the immutable blockchain protocol Γ satisfies common prefix property, then Γ' satisfies the redactable common prefix property.*

Proof. Note that if there is no edit in the chain C, then Γ' behaves exactly like the immutable blockchain protocol Γ, 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_2, which is later edited by honest P_2. We can observe that the adversary cannot propose another candidate $\widetilde{B_j^*} \neq B_j^*$ such that $H(\widetilde{B_j^*}) = H(B_j^*)$, 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 μ, where μ denotes the fraction of stakes controlled by the adversary. We prove that Γ' 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 Γ satisfies the chain quality property with parameters (k, μ), then Γ' satisfies the chain quality property with parameters (k, μ).*

Proof. Note that if there is no edit in the chain, then Γ' behaves exactly like the immutable blockchain protocol Γ, 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 dada), 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 adaptive corruption 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 $2/3 \cdot T$, 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 $2/3 \cdot T$ 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 $1/2^{hashlen}$.

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 $2/3 \cdot T$, 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 $2/3 \cdot T$ 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 σ from U_i 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.

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 $2/3 \cdot T$ users on the adversarial edit request message, however, among the $2/3 \cdot T$ 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 $2/3 \cdot T$ 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 $2/3 \cdot T$ 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 singing 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 Γ satisfies the chain growth property with parameters (τ, s), then Γ' also satisfies the chain growth property with parameters (τ, s) under the editing policy $\mathcal{P}$.*

Proof. Note that any edit operation in Γ' 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 Γ' preserve the chain growth property of Γ. □

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_{vr}, 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 $|H|$, $|\pi|$ 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 1: Experimental cost of each operation (ms)

	t_{pr}	t_{vr}	t_s	t_v	t_a
128-bit	0.46	1.52	0.49	1.52	0.002
192-bit	0.68	5.29	0.85	5.29	0.003

Table 2: Experimental size of each element (bits)

	$\lvert H\rvert$	$\lvert\pi\rvert$	$\lvert msig\rvert$
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 $|msig| + |PROOF| = |msig| + 2/3 \cdot T(|H| + |\pi|)$, 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 *misg*, which costs about $2/3 \cdot T \cdot t_{\mathrm{vr}}$, $2/3 \cdot T \cdot t_{\mathrm{v}}$ and $2/3 \cdot T \cdot t_{\mathrm{a}}$, 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 denotes 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 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.

6 DISCUSSION AND MULTIPLE REDACTIONS

In the 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 redactable 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 $B_j = (sj_j, st_j, d_j, ib_j, B_{\pi,j}, \sigma_j)$, 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 $B_j^* = (sj_j, st_j, d_j^*, ib_j^*, B_{\pi,j}, \sigma_j)$. 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.

procedure validateBlockExt(B)

Parse $B = (sl, st, d, ib, B_\pi, \sigma)$;
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 B_π including the verification key vk of leader,
  if invalid **return** 0;
if the signature σ on $(sl, G(st, d), ib, B_\pi)$ or
  on $(sl, ib^{(1)}, ib^{(1)}, B_\pi)$ is verified with vk
  then return 1;
else return 0.

Algorithm 6: The extended block validation algorithm

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.

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 $B_j = (sl_j, st_j, d_j, ib_j, B_{\pi,j}, \sigma_j)$;
  parse $B_{j-1} = (sl_{j-1}, st_{j-1}, d_{j-1}, ib_{j-1}, B_{\pi,j-1}, \sigma_{j-1})$;
  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 $st_j = H(sl_{j-1}, G(st_{j-1}, d_{j-1}), ib_{j-1})$ **then**
    $j = j - 1$;
  else if $st_j = H(sl_{j-1}, ib_{j-1}^{(1)}, ib_{j-1}^{(1)})$ and $\mathcal{P}(C, B_{j-1}) = 1$
    then $j = j - 1$;
  else return 0;
return 1.

Algorithm 7: The extended chain validation algorithm

Validating candidate editing block. To validate a candidate editing block, the users run validateCandExt algorithm (Algorithm 8). 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.

The checkVote algorithm (Algorithm 4) for checking voting right and collectVote algorithm (Algorithm 5) for collecting votes remain unchanged.

procedure validateCandExt(C, B_j^*)

Parse $B_j^* = (sl_j, st_j, d_j^*, ib_j, B_{\pi,j}, \sigma_j)$;
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 $B_{j-1} = (sl_{j-1}, st_{j-1}, d_{j-1}, ib_{j-1}, B_{\pi,j-1}, \sigma_{j-1})$;
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 $B_{j+1} = (sl_{j+1}, st_{j+1}, d_{j+1}, ib_{j+1}, B_{\pi,j+1}, \sigma_{j+1})$;
if $st_j \neq H(sl_{j-1}, ib_{j-1}^{(1)}, ib_{j-1}^{(1)})$ or $st_{j+1} \neq H(sl_j, ib_j^{(1)}, ib_j^{(1)})$
  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 8: The extended candidate block validation algorithm

REFERENCES

[1] Akasha. https://akasha.world.

[2] Ethereum project. https://www.ethereum.org/.

[3] The illinois blockchain initiative. https://illinoisblockchain.tech.

[4] Rewritable blockchain. May 8 2018, uS Patent 9,967,096.

[5] Steem. https://steem.

[6] 2016. Accenture files patent for editable blockchain. https://tinyurl.com/yblq9zdp.

[7] 2017. Governments may be big backers of the blockchain. In *The Economist*. https://goo.gl/uEjckp.

[8] 2018. The EU general data protection regulation. https://eugdpr.org/the-regulation/.

[9] 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.

[10] 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.

[11] Razvan Barbulescu and Sylvain Duquesne. 2017. Updating key size estimations for pairings. *Journal of Cryptology* (2017), 1–39.

[12] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *ASIACRYPT 2018*, Vol. 11273. Springer, 435–464.

[13] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. *ASIACRYPT* (2001), 514–532.

[14] 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.

[15] CBinsights. 2018. Banking is only the beginning: 50 big industries blockchain could transform. https://www.cbinsights.com/research/ industries-disrupted-blockchain/.

[16] Jing Chen and Silvio Micali. 2017. Algorand. In *arXiv:1607.01341v9*.

[17] CPhil Daian, Rafael Pass, and Elaine Shi. 2019. Snow white: robustly reconfigurable consensus and applications to provably secure proof of stake. In *Proceedings of Financial Cryptography and Data Security*.

[18] 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.

[19] 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*.

[20] Dominic Deuber, Bernardo Magriy, Sri Aravinda, and Thyagarajan Krishnan. 2019. Redactable blockchain in the permissionless setting. In *IEEE Symposium on Security and Privacy 2019*.

[21] 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.

[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] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The bitcoin backbone protocol: Analysis and applications. 9057 (2015), 281–310.

[24] 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.

[25] S. Goldwasser, S Micali, and R.L. Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* 17 (1988), 281–308.

[26] 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.

[27] 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/.

[28] Christoph Jentzsch. Decentralized autonomous organization to automate governance. https://download.slock.it/public/DAO/WhitePaper.pdf.

[29] 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.

[30] J. Mathew. 2015. Bitcoin: Blockchain could become 'safe haven' for hosting child sexual abuse images. http://www.dailydot.com/business/bitcoinchild-porn-transaction-code/.

[31] S. Micali, K. Ohta, and L. Reyzin. 2001. Accountable-subgroup multisignatures: Extended abstract. In *8th Conference on Computer and Communications Security*. ACM, 245–254.

[32] S. Micali, M. O. Rabin, and S. P. Vadhan. 1999. Verifiable random functions. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 120–130.

[33] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.

[34] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT 2017*, Vol. 10211. Springer, 643–673.

[35] Ivan Puddu, Alexandra Dmitrienko, and Srdjan Capkun. 2017. μ chain: How to forget without hard forks. In *IACR Cryptology ePrint Archive, 2017/106*.

[36] K. 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.

[37] 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.

[38] G. Tziakouris. 2018. Cryptocurrenciesąła forensic challenge or opportunity for law enforcement? an interpol perspective. *IEEE Security & Privacy* 16 (2018), 92–94.