This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

PRBFPT: A Practical Redactable Blockchain Framework with a Public Trapdoor

Weiqi Dai, Jinkai Liu, Yang Zhou, Kim-Kwang Raymond Choo, Xia Xie, Deqing Zou, Hai Jin

***Abstract*—While blockchain is known to support open and transparent data exchange, partly due to its nontamperability property, it can also be (ab)used to facilitate the spreading of fake and misleading information or information that was subsequently discredited. Hence, this paper proposes a practical, redactable blockchain framework with a public trapdoor (hereafter referred to as PRBFPT). PRBFPT comprises an editing scheme for adding blocks using a new type of blockchain with a chameleon hash. Specifically, PRBFPT is able to involve all nodes in the blockchain in the editing operations by means of a public trapdoor, without requiring additional trapdoor management by predefined nodes or organizations. PRBFPT is also designed to audit and record the content of each editing operation. In other words, after editing and deleting the original data, PRBFPT can still verify its legitimacy. We also propose a contract-based locked voting scheme to better support voting. We then evaluate the prototype implementation of PRBFPT, whose findings show that the total time consumption of adding modules is at the millisecond level, with a negligible impact on the performance of the original system. In addition, the evaluation findings show that the cost of initiating the special transactions is comparable to the consumption of normal Ethereum transactions and is within a manageable range.**

***Index Terms*—Redactable blockchain, Rumors, Chameleon hash, Publicly trapdoor, Smart contract.**

I. Introduction

BLOCKCHAIN is known for its ability to ensure data openness, transparency and distributed consistency [1], [2], but these features can also be exploited to spread rumors, including misleading and malicious information [3], [4], with potentially devastating consequences (e.g., social panic). Due to the tamper-evident nature of blockchain, we can quickly locate the virtual identity of the originator of the rumor. However, rumors that have been recorded in the blockchain cannot be removed. In response, conventional blockchain systems can deploy a scheme similar to database version updates to delete or overwrite data on the chain. Specifically, a new version of the data is created and invalidated by pointing to the old version when it is updated in the blockchain ledger. In practice, the data are not deleted since they are still retrievable. Such practices also do not comply with privacy legislation such as the European Union's General Data Protection Regulation (GDPR), which states that

> The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay where one of the following grounds applies [...][1]

This point reinforces the importance of designing blockchain systems that are secure and auditable and yet have the ability to completely delete data. In 2017, for example, Ateniese et al. [5] proposed a redactable blockchain that supports the deletion of data in the blockchain ledger directly (rather than only supporting pseudodeletion, as in the case of database update logs). The core building block of this approach is the chameleon hash [6]–[12], which is a hash function in which the hash is parameterized by the public key hk. As long as the trapdoor (corresponding to the key tk of hk) is not known, it is collision-resistant, as is the normal hash function. In contrast, if the trapdoor tk is known, then hash collisions can be easily created. Thus, nonleakable trapdoors form the basis for the security of such redactable blockchains. The chameleon hash has been widely used in blockchain; for example, Kumar et al. [13] used the chameleon hash to support data privacy protection.

Subsequent approaches improved the protection of trapdoors at the algorithmic level and provided additional properties for redactable blockchains. For example, redactable blockchains built on the chameleon hash and attribute-based encryption [14], [15] proposed by Derler et al. [16] support fine-grained modifications at the transaction level with a finer granularity of editing. Duan et al. [17] considered adding policies for the use of the chameleon hash to avoid misuse and designed a policy-based chameleon hash with black-box traceability. Policy-based approaches can be effective, but policies may change over time and differ between situations. Hence, Xu et al. [18] proposed a policy-revocable chameleon hash scheme. Jia et al. [19] inserted the encoding of redacted blocks into accumulators for more efficient edit traceability and proposed

This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB1006000, in part by the National Natural Science Foundation of China under Grant 62072202. The work of K.-K. R. Choo was supported only by the Cloud Technology Endowed Professorship. (Corresponding author: Deqing Zou.)

W. Dai, J. Liu, Y. Zhou, and D. Zou are with School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: wqdai@hust.edu.cn; liujink@hust.edu.cn; zhouyang_cse@hust.edu.cn; deqingzou@hust.edu.cn).

K.-K. R. Choo is with the Department of Information Systems and Cyber Security and the Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX 78249-0631 USA (e-mail: raymond.choo@fulbrightmail.org).

X. Xie is with School of Computer Science and Technology, Hainan University, HaiKou 570100, china (e-mail: shelicy@hainanu.edu.cn).

H. Jin is with School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: hjin@hust.edu.cn).

[1]https://gdpr.eu/article-17-right-to-be-forgotten/, last accessed November 30, 2023

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

a decentralized chameleon hash to guarantee that redactability does not destroy the decentralized nature of the blockchain. The security of the redacting scheme relies on the privacy of the chameleon hash trapdoor. In [20], the approach of [16] was enhanced with the ability to control trapdoor permissions and allow auditing of the entity alleged to have exploited the trapdoor to make changes to the data. In [21], the authors combined zero-knowledge proofs [22], [23] with the chameleon hash to enhance collision resistance while reducing the size of chameleon hash functions. In [24], the lattice signature [25] was combined with a chameleon hash to enhance the quantum resistance of trapdoor protection. Li et al. [26] proposed a tagged chameleon hash for postquantum security without zero-knowledge proofs and proved its security under a standard model. Some schemes focus more on other areas; e.g., Li et al. [27] decoupled the voting phase from the consensus layer to achieve instantaneous redaction.

To perform trapdoor secrecy and management, however, the redactable blockchains discussed in the preceding paragraphs have complex algorithms that limit their deployment. For example, Ateniese et al.'s approach [5] requires trapdoor sharing among trusted groups using MPC techniques [28]–[30], the approaches in both Derler et al. [16] and Tian et al. [20] rely on trusted third parties for attribute distribution to grant another entity permission to modify the blockchain, and Wu et al.'s approach [24] also greatly increases the complexity of the chameleon hash algorithm. Moreover, these approaches always require one or more trusted entities to edit the data, rather than allowing all users to participate in the editing operation together. When messages are deleted using these approaches, they hardly leave a trace of the original data. These limitations are potential barriers to implementation; for example, in the absence of the deliberate retention of copies, if data need to be restored as evidence, the entity redacting the data can fake its credibility by forging the original data. In public chains (e.g., Bitcoin [31] and Ethereum [32]), where nodes come and go arbitrarily, it is difficult to elect such a group with the privileges of most users.

In a separate line of inquiry, Deuber et al. [33] proposed using consensus voting to support transaction-level rewriting and enhanced user awareness of editing operations. However, such an approach can potentially cause a "chain break" in the blockchain during the editing of blocks. Moreover, this approach requires the collection of sufficient votes, and the confirmation period for voting is too long (i.e., not suitable for time-sensitive applications). The approach also expects miners not to adopt a default strategy of voting yes or no.

A. Our contributions

To address the challenges associated with secure and auditable editing and chameleon hashing trapdoor management on the blockchain, we propose a practical redactable blockchain scheme based on a new block structure and voting with the benefits of no need for trapdoor management, fine-grained redacting of transactions, and auditability. The contributions of this paper are as follows:

- PRBFPT, a redactable blockchain scheme, is implemented using a new block structure, smart contract voting and a chameleon hash. We propose a new blockchain structure that achieves editability based on this structure. The redactable scheme has two steps, namely, voting and editing. A new block header structure and insertion block approach are used to implement editing operations for transactions in a block. The editing operation is performed by all nodes rather than selected nodes or other groups. The editing operation is auditable and has the option of deleting the original block data associated with the edit operation. The length of the edited data is not limited by the length of the original transaction data. There is no need for specific chameleon hashes or management of their trapdoors throughout the implementation of the scheme. The success of an editing operation is determined by the results of the associated vote.
- To implement the scheme more efficiently, we also propose a voting scheme based on contractual locking, which allows users associated with the content of the vote to participate in the vote instead of all or specified users.

In the context of blockchain nodes, we design the block editing process to be an operation that all nodes must complete synchronously, similar to a normal transaction.

Table I comparatively summarizes our solution with other redactable blockchain solutions, and one can observe that the most notable features of our designed solution are the redesigned block structure and the use of voting protocols to achieve a redactable blockchain. Unlike other studies that focus on modifying the chameleon hash to avoid various security issues, our redactable blockchain scheme has security that reduces to the security of the blockchain itself. Therefore, our scheme is not restricted to a specific chameleon hash and can use any chameleon hash scheme with desired properties, such as the quantum-secure chameleon hash, that satisfies the definition of the chameleon hash in this paper. In addition, there is no need for additional protection of trapdoors; complex trapdoor protection methods are an important reason for the inefficiency of a scheme. Some schemes need to introduce third parties to satisfy their chameleon hash properties, which potentially erodes the decentralized nature of the blockchain. Editing operations can have a significant impact on the blockchain, so auditability is also a concern.

B. Outline

The rest of this paper is organized as follows. In the next section, we will describe our changes to the blockchain structure. The third section will describe the design of our proposed approach. The fifth section will present our prototype implementation on Ethereum, as well as the evaluation metrics and findings. The last section concludes this paper.

II. System Design

In this section, we describe the core process of one edit and some important changes to the basic structure of the blockchain.

A summary of the notation used in this paper is outlined in Table II.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

TABLE I: Differences between PRBFPT and other approaches

Approach	Technology†	Auditable	Additional protection for trapdoors	Extra authority
Ours	New Block Structure, Voting	Yes	No	No
Ateniese et al. [5]	CH, MPC	Yes	No	Yes
Derler et al. [16]	CH, CP-ABE	No	Yes	Yes
Wu et al. [24]	CH, Lattice	Yes	Yes	No
Jia et al. [19]	CH, MPC	Yes	Yes	No
Duan et al. [17]	CH, FAME	Yes	Yes	Yes
Li et al. [26]	CH, Lattice	Yes	Yes	No

† Voting stands for smart contract voting, CH stands for chameleon hash, MPC stands for secure multi-party computation, CP-ABE stands for ciphertext-policy attribute-based encryption, and FAME stands for fast attribute-based message encryption.

TABLE II: Summary of Notation

Notation	Description
$\mathcal{CH}$	Chameleon hash algorithm
(hk, tk)	Public and private keys for the chameleon hash algorithm; tk is also known as a trapdoor
κ	The security level of the chameleon hash algorithm parameters
$\mathsf{H}(\cdot)$	Cryptographically secure hash algorithm such as SHA-256
m	Input plaintext string for the hash algorithm
h	The result of a string going through the chameleon hash process, usually denoted as $h = \mathcal{CH}(m, r)$
$\mathcal{B}_i$	The i-th block on the blockchain
$\mathcal{B}_i^*$	The edited block of the i-th block on the blockchain
τ	A transaction in a blockchain containing domains $Data, RTrans$, etc.
A	$A = \mathcal{CH}(\mathsf{H}(\tau.Data), \tau.RTrans)$

A. Overview

As previously discussed, we aim to implement secure editing of the blockchain without introducing any additional entities (i.e., only the blockchain and its users are involved in the system model). Our core idea is to first use the blockchain to vote on the editing operation of the offending data and then let all nodes execute the editing module of the added block to ensure the consistency of the blockchain. In our proposed new blockchain structure, the voting-based approach can minimize the security risk after the exposure of the chameleon hash algorithm trapdoor and make the editing operation auditable. The workflow for content removal and modification in PRBFPT is shown in Fig. 1 and explained below.

1) **Vote to identify offending content.** An initiator makes a request to edit content τ and provides the edited content τ'. A voting process that can be recorded in the blockchain determines whether the content should be edited or not. Upon the conclusion of the process, relevant records are left in the blockchain for the nodes to authenticate the validity of the process.
2) **Permission validation with a block addition edit.** After receiving the validation transaction and successful verification of the validity of the voting process, all nodes perform the ledger update separately according to the predefined steps. First, a new block $\mathcal{B}^*$ is made. Then, $\mathcal{B}^*$ replaces the offending data τ in the old block $\mathcal{B}$ with data τ' that have been validated by voting. Other than that, the data in the two blocks are not different. Subsequently, the node will place $\mathcal{B}^*$ before $\mathcal{B}$ and use chameleon hashing so that the chain does not break. Finally, $\mathcal{B}$ will be labeled. In this system, the block data of $\mathcal{B}$ are no longer of any use other than for verification.

The block edit proposer and the voting participants can be any users in the blockchain. The user who finds the offending data can propose editing the blockchain and set the corresponding range of voting participants according to the modification requirements, which are by no means limited to specific users.

From a practical perspective, our scheme assumes that an adversary can eavesdrop on any message in the blockchain network and can control multiple blockchain nodes but cannot disrupt the normal consensus of the blockchain or force a fork of the blockchain.

B. Chameleon hash function

We define the chameleon hash function $\mathcal{CH}$ as a tuple $(ChamPre, ChamHash, ChamVer, ChamCol)$. These functions are used in PRBFPT for the chameleon hash calculation of both the block header and transaction. The definition is as follows:

- $(hk, tk) \leftarrow ChamPre(\kappa)$: The secret key (trapdoor) generation algorithm takes a security parameter κ as input and outputs a public key hk and a trapdoor secret key tk. In PRBFPT, hk and tk are not changed. All nodes and all blocks use the same hk and tk.
- $h \leftarrow ChamHashCal(hk, (m, r))$: The chameleon hash computation algorithm takes a chameleon hash public key hk, a chameleon hash trapdoor private key hk, a plaintext string m, and a verified random number R as input and outputs a chameleon hash value h. To simplify the description, we use a representation such as $h = \mathcal{CH}(m, r)$.
- $\{0, 1\} = ChamVer(hk, (m, r), h)$: The chameleon hash verification algorithm takes as input a chameleon hash public key hk, a chameleon hash trapdoor private key hk, a plaintext string m, a verified random number r and a chameleon hash h. If the hash h is a pair of a plaintext and a verified random number (m, r) of a valid hash, then the algorithm will output 1 (otherwise, it will output 0).
- $r' \leftarrow ChamCol(hk, tk, m', (m, r))$: The chameleon hash collision generation algorithm takes as input a

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

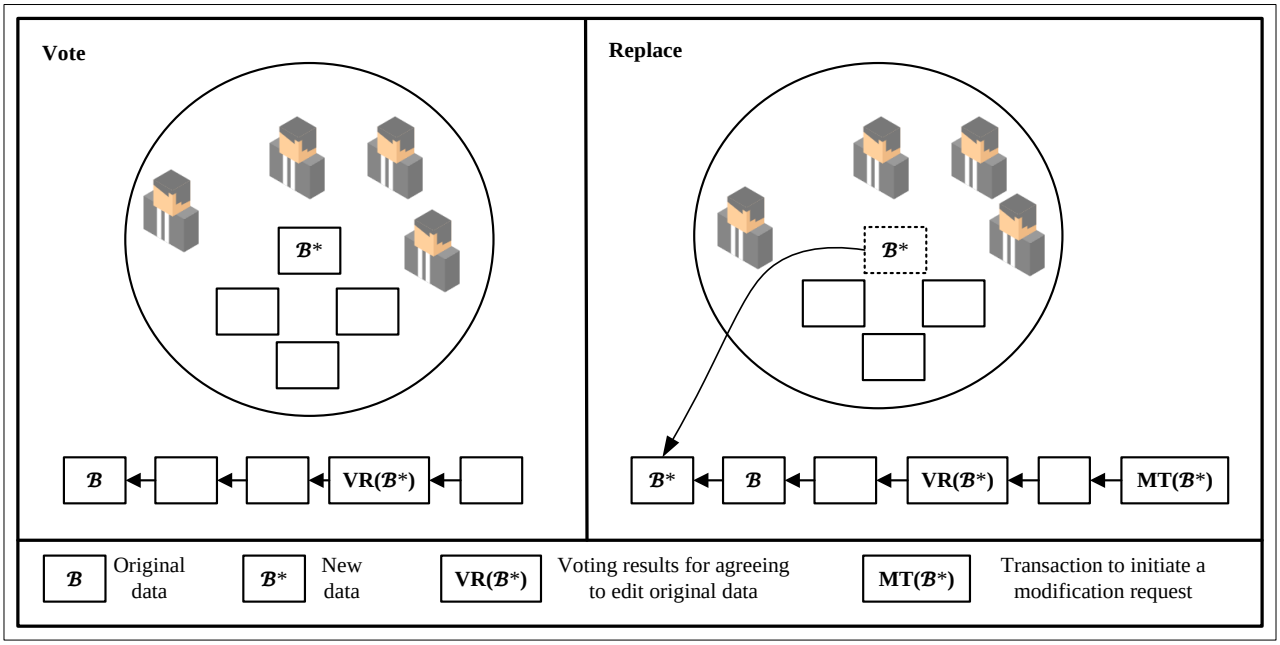


Fig. 1: Workflow for a simplified repair

chameleon hash trapdoor tk, a plaintext string m, a verified random number r and a new plaintext string m' and outputs a verified random number r' that matches the plaintext string m', which verifies the random number r' by checking that $ChamVer(hk,(m,r),h) = ChamVer(hk,(m',r'),h) = 1$. If the plaintext with the verified random number (m,r) is invalid, then $null$ will be output.

C. New block structure

Changes to the block structure are necessary to eliminate the security risks associated with exposed trapdoors. The block structure necessary for PRBFPT is described below.

New block header field: To easily obtain the newly added blocks and old blocks related to editing operations, as well as to maintain the connectivity of the blockchain even after the addition of blocks, we added the following fields to the block header structure:

- $PreBlock$, $NextBlock$. These are the indices that are used to find the previous block and the next block of the current block. After inserting a block by adding a block at the end of the ledger, the order of the blocks is not the same as the order in the ledger. When performing block validation, using the index makes it easy to find the desired block.
- $RHeader$. This is a required field for the block header hash calculation. PRBFPT specifies that the value of $RHeader$ is 0 when a new block is generated. In other words, if the $RHeader$ of a block is not 0 (only the method $ChamCol$ can generate a new $RHeader$), this proves that it has been edited. In PRBFPT, its data are treated as invalid, and it can only be used to verify the validity of adding blocks.
- $ChamParam(hk,tk)$. This is the set of public and private key parameters for the chameleon hash function. It includes the chameleon-hash public key hk and the trapdoor tk. The specific generation algorithm and the process involved in the computation will be described in detail in the Hash Calculation section. $RHeader$ and $PreCHash$ are necessary fields for this operation.
- $PreCHash$. The $PreCHash$ is the key to maintaining the connectivity of the blockchain. It is the result of a calculation using the method $ChamHashCal$, the hash result of the previous block $PreHash$ and $RHeader$. The calculation equation is $PreCHash \leftarrow ChamHashCal(hk,(PreHash,RHeader))$. To obtain the $PreHash$ of a new block, the result of $PreCHash$ will not change by updating $RHeader$ via the function $ChamCol$. This allows a new block to be inserted before a block without breaking the chain connectivity.

New Transaction Fields: To implement transaction-level data editing, we modified the Merkle Hash generation scheme. The following are the necessary additions to the transaction fields:

- $RTrans$. This is the key field used for the Merkle hash calculation. PRBFPT specifies that the value of $RTrans$ is 0 when new transaction data are generated. In other words, if the $RTrans$ of a transaction is not 0 (only the function $ChamCol$ can generate a new $RTrans$), it is proven to have been edited. In PRBFPT, this type of transaction usually has no data. If it has data, these data can only be treated as a sample of preedited data and used for auditing.
- $SigTrans$. This is the flag of the transaction. It is used to distinguish edited transactions from normal transactions. It can be used to identify malicious modifications by nodes with the algorithm 2.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

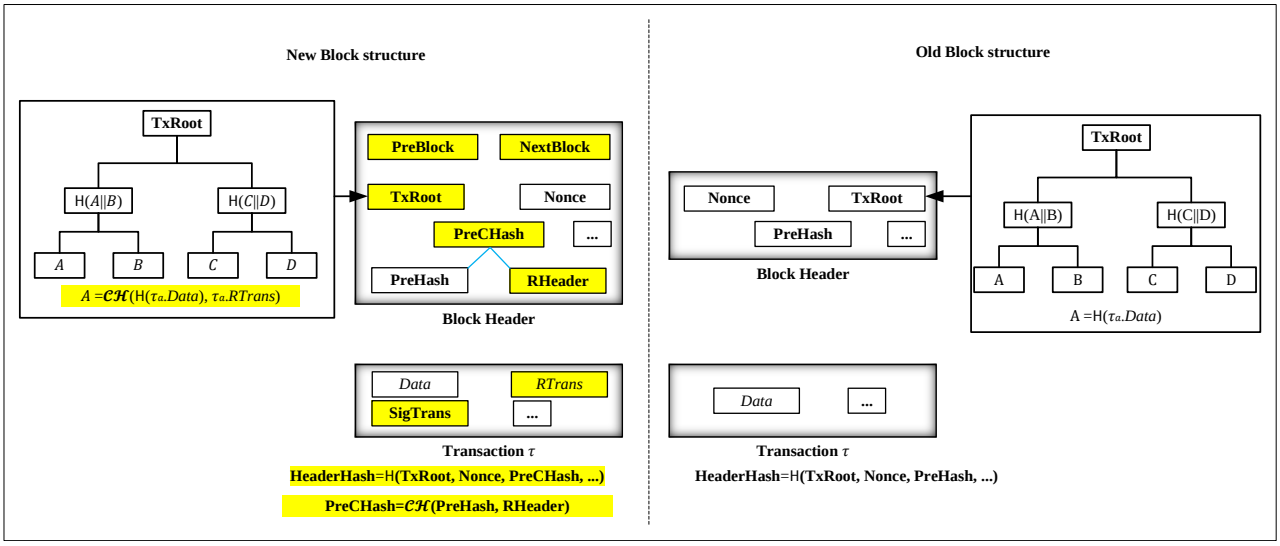


Fig. 2: Comparison of data computations for the block header

D. Hash Calculation

1) Merkel hash calculation: $TxRoot \leftarrow MerkelCal(\tau_a, \tau_b, \tau_c, ...)$. This is the new Merkle hash calculated according to the traditional Merkle hash calculation method. SHA-256 is generally used as the main function for the calculation, followed by the use of $\mathsf{H}(m)$ to stand in for the SHA-256 computation on string m. Unlike the traditional Merkle hash, which is directly calculated by using the transaction hash, PRBFPT uses the chameleon hash of the transaction hash for the Merkle hash calculation. This enables transaction-level block data editing. Suppose the data of transaction τ_a are $\tau_a.Data$ and the number of transaction validations is $\tau_a.RTrans$. Then, the chameleon hash calculation equation for A is $A \leftarrow \mathcal{CH}(hk, \mathsf{H}(\tau_a.Data), \tau_a.RTrans)$. The $RTrans$ values of transactions that are not edited are 0. hk and tk are fixed. Therefore, A can only be computed by $\tau_a.Data$. Obviously, for transactions that have not been edited, the security of the new Merkle hash is no different from that of the traditional blockchain. The specific calculation is shown in Fig. 2.

2) Block header hash calculation: $HeaderHash \leftarrow HeaderCal(TxRoot, PreCHash, Nonce, ...)$. This is the hash calculation algorithm, whose calculation process is consistent with the traditional block header hash calculation. Slightly different from the traditional calculation, PRBFPT calculates $PreCHash$ first and then uses $PreCHash$ for the block header hash calculation. For blocks that are not edited, $RHeader$ is 0, and the $ChamParam(hk, tk)$ values associated with the computation process are public and consistent. Therefore, the result of the chameleon computation does not change due to trapdoor exposure. This means that for blocks where $RHeader$ is 0, the security is no different from that of traditional blocks.

III. Block Editing Scheme with a Public Trapdoor

In this section, we explain PRBFPT's design in terms of voting to identify the offending content and the editing scheme for adding blocks. We also provide a validation algorithm to verify the edited blocks. The final part of the section discusses PRBFPT's improvements to trapdoor management, features of editing operations, and some possible security risks.

A. Block and transaction type definitions

To facilitate the understanding of PRBFPT, we give some definitions.

- **Secure block, secure transaction.** A secure block or secure transaction is a block or transaction that has not been edited. The value of $RHeader$ in the block header of a secure block is initially set and unaltered, indicating that it is unedited. It can be inferred from the two algorithms of the block header hash calculation and Merkle hash calculation that its security is consistent with that of traditional blocks. The function of the secure block is consistent with that of the traditional block. The $RTrans$ of secure transactions is identical to the $RHeader$ of secure blocks, and the security and functionality are also equivalent to those of traditional blockchain.
- **Invalid block, invalid transaction.** If the $RHeader$ of a block is not the initial value, we consider it to be an invalid block. The $RTrans$ of an invalid transaction is the same as the $RHeader$ of an invalid block. The contents of the invalid block are only used for the validation of unedited transactions. Invalid transactions are only used for validation of the original transaction copy. Invalid blocks are generated only by edit operations, and invalid transactions exist only in invalid blocks.

The two transactions introduced in the framework are described next.

- $VotingRes$**.** $VotingRes$ is the validation result generated by the voting policy when an edit proposal passes the voting policy. This result is recorded in the blockchain and used for the validation of the edit operation.
- $ModifiedTrans$**.** After $VotingRes$ is recorded in the blockchain ledger, the originator of the edit

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

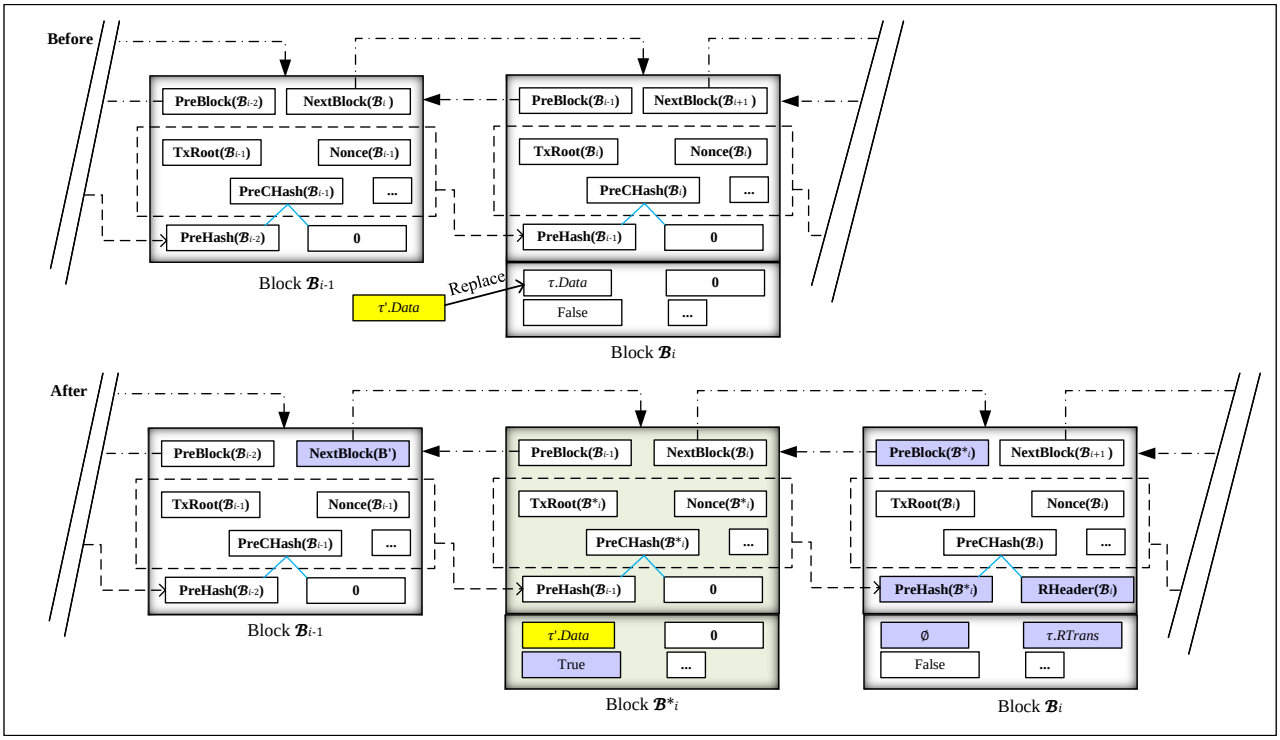


Fig. 3: Editing scheme for adding blocks

proposal sends a modification request transaction $ModifiedTrans$. This transaction is used by the node to automate the editing of the ledger and contains the necessary information, such as the on-chain index of the offending content and the edited content.

B. Vote to determine the content of the violation

The identification of the offending content is actually entirely dependent on the requirements of the application. We will not set the requirements for specific content here but will only analyze how to make the determination. We will discuss this for three types of blockchains (private, consortium and public). Finally, we will give a violation content identification procedure, called the contract-based locked voting scheme. The details are discussed below.

- **Private blockchain.** The bookkeeping permissions of private blockchains are often held by a central authority, so the determination of editing content can also be given to this authority.
- **Consortium blockchain.** The bookkeeping permissions of consortium blockchains (e.g., Fabric [34]) are held by a set of already determined nodes, so the determination of the editorial content is simply left to such nodes for voting. Of course, it is also possible to adopt the voting scheme using contracts that we proposed in the public chain section for editorial content determination.
- **Public blockchain.** The uplinking authority of the public blockchain is mastered by all the nodes involved in the public blockchain, and there are already some options, which we will discuss later.
  - **Voting by all nodes in the public blockchain.** A permission determination mechanism whereby miners vote when packing blocks is described in Deuber et al.'s paper, and a block editing function is implemented based on this vote. In this work, no chameleon hash is used, and the blockchain will "break" when editing the blocks. Moreover, the scheme requires every miner to participate in voting. In practice, most miners may not care about editing content and tend to adopt a default yes or no strategy. This can make the usability of the scheme questionable. However, it is certainly an excellent solution in terms of design ideas and can be well integrated with our design.
  - **Voting by the identified partial nodes in the public blockchain or introducing trusted third parties.** A paper by Ateniese et al. [5] mentions selecting trusted partial nodes in the public blockchain and granting editing privileges; a trusted third party can also be used to grant editing privileges, as in the work of Deler et al. Obviously, these approaches have some limitations for use in public blockchains.

Contract-based locked voting scheme. Based on the existing schemes, we consider that editorial content determination should be proposed by content stakeholders. We propose a contract-based locked voting scheme with the help of smart contracts [35] and transaction identifier (TXID) technology (the TXID can be used to index the corresponding transaction in the blockchain). The results of this scheme can be easily authenticated by nodes and can be easily made compatible with blockchains that have smart contract functionality. Then,

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

the default voting scheme used is this scheme.

* **Member record.** When each member uses the contract for the first time, the contract records the member's address and adds it to the member set $AddSet$. The contract records the number of times the member invokes the contract and the total number of invocations. We evaluate the member's activity according to these numbers.
* **Set the voting period and strategy.** Voting cannot be conducted endlessly. Referring to the concurrency speed of Ethereum[2] and assuming the minimum time that a voting event may be noticed as a hotspot [36] [37], we generally set the voting period (the time limit of one vote) to two or three days. The setting of the voting strategy depends on the specific application. For example, voting success can be defined as the number of supporters outweighing the number of opponents.
* **Voting stage.** When a vote is initiated, we lock the current $AddSet$ and sort it in descending order of member activity. A new set $VoterSet$ is formed by selecting highly active users whose cumulative activity is more than half of the total activity. Only the members in $VoterSet$ corresponding to each voting event can vote on that event. We use the contract for voting and have a voting period predefined in the contract. After a certain number of members agree to vote in a period, we will obtain a tuple $VotingRes(OldHash, NewDataHash, Type, Res)$ recorded on the chain. $OldHash$ represents the hash of the transaction to be modified. $NewDataHash$ denotes the edited data hash. $Type$ denotes the type of editing, such as delete, modify, etc. Res indicates the result of the vote. This result has a value of $true$ when the voting policy is satisfied; otherwise, it is $false$. $VotingRes$ is typically generated automatically by the contract or by a validation request. We specify that none of the $VotingRes$ values can be edited, which can be easily verified. The meta-ancestor can be found in the blockchain via the TXID of $VotingRes$ ($TxidVres$).
* **VotingResults.** A node indexed by $TxidVres$ to the validation result $VotingRes$ generated during the voting phase. This confirms that $VotingRes$ has not been edited. The node verifies that the accepted $NewData, Type$ is correct and Res is $true$ to determine the validity of the vote.

C. Editing scheme for adding blocks

We designed a ledger editing process that is executed by all nodes. The nodes execute this process automatically after the voting result $VotingRes$ above is uploaded and the node receives the modification request transaction $ModifiedTrans$. The scenario described below is a one-time edit process for a particular violation record.

To facilitate understanding, we add some definitions.

- $\mathcal{B}_i$ is the block containing the transaction that needs to be edited, and $\mathcal{B}_{i-1}$ is the block that precedes it.
- τ is the original transaction that will be edited, which is recorded in $\mathcal{B}_i$. τ' is the edited transaction.
- $ModifiedTrans(NewData, Type, TxidVres)$. This is the modification request transaction sent by the editor. $NewData$ represents the edited data, $Type$ represents the edit type, and $TxidVres$ is used to index the voting result corresponding to $NewData$.

The details of the process are explained below.

1) **Verify** $VotingRes$**.** After the node receives $ModifiedTrans$, it finds the voting result $VotingRes$ according to $TxidVres$ and verifies that $NewData, Type$ matches the record in $VotingRes$.
2) **Create new block data.** Create a new block $\mathcal{B}_i^*$ whose block data contain all the transactions in $\mathcal{B}_i$'s data except τ and τ'. In other words, the data of $\mathcal{B}_i^*$ are the data of $\mathcal{B}_i$ that have been replaced by τ.
3) **Fill in a new block header.** Fill in the block header of $\mathcal{B}_i^*$ based on the block header of $\mathcal{B}_i$ and the block data of $\mathcal{B}_i^*$. Compute the hash $\mathsf{H}(\mathcal{B}_i^*)$ of $\mathcal{B}_i^*$.
4) **Insert new block.** Insert block $\mathcal{B}_i^*$ before $\mathcal{B}_i$, remove the information about transaction τ from $\mathcal{B}_i$, and compute the hash collision of transaction τ (compute $RTrans$) to ensure that the Merkle hash of $\mathcal{B}_i$ remains unchanged. After this, populate $\mathsf{H}(\mathcal{B}_i^*)$ with the previous block hash field in the $\mathcal{B}_i$ block header and recompute the hash collision of $\mathcal{B}_i$ (compute $RHeader$) to ensure that the hash $\mathsf{H}(\mathcal{B}_i)$ computed by $\mathcal{B}_i$ remains unchanged. Eventually, fields such as the front and back indices of blocks $\mathcal{B}_{i-1}, \mathcal{B}_i^*, \mathcal{B}_i$ are modified. At this point, $\mathcal{B}_i$ is marked as an invalid block. Fig. 3 shows the change in the block header fields in block $(\mathcal{B}_{i-1}, \mathcal{B}_i^*, \mathcal{B}_i)$ after inserting the new block.
5) **Update ledger.** Write the constructed new block to the ledger as an add. After that, construct block $\mathcal{S}$ by following the normal block construction process for $ModifiedTrans$ and the transactions packaged with it. It is placed at the very end of the ledger in the same way as the additions. At this point, from the perspective of the ledger, the blocks $(\mathcal{B}_{i-1}, \mathcal{B}_i^*, \mathcal{B}_i, ..., \mathcal{S})$ are placed in the order of $(\mathcal{B}_{i-1}, \mathcal{B}_i, ..., \mathcal{B}_i^*, \mathcal{S})$. Fig. 4 shows the relationship between the positions of these blocks in the ledger.

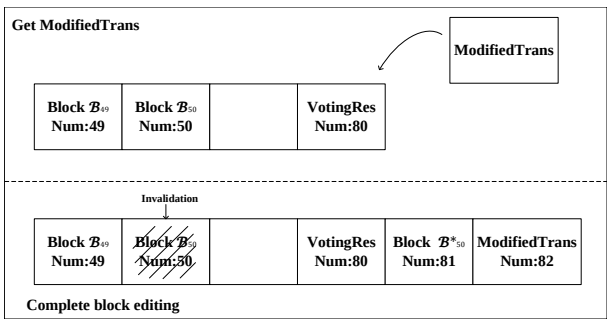


Fig. 4: Changes to the ledger after adding blocks

[2]https://ethereum.org/en/developers/docs/blocks/

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

D. Transaction validation algorithm

Before introducing the transaction validation algorithm, the following definitions are added to facilitate the understanding of the flow of algorithm execution.

- $ModifiedTrans(NewData, TxidVres)$: This is the modified request transaction, where $NewData$ represents the edited data and $TxidVres$ is used to locate the voting results corresponding to $NewData$.
- $VotingRes(OldHash, NewDataHash, Res)$: This is the voting result validation transaction, where $OldHash$ represents the data hash of the transaction to be edited. $NewDataHash$ denotes the edited data hash. Res denotes the result of the vote. This result has the value $true$ when the voting policy is satisfied; otherwise, it is $false$.
- $\tau(Data, SigTrans, RTrans)$: The structure of a transaction. $Data$ is the data for the transaction. The $Data$ of a particular transaction can be transformed into structures such as $ModifiedTrans$ and $VotingRes$. $SigTrans$ for ordinary and voting transactions is $false$, and $SigTrans$ for edited transactions is $true$. $RTrans$ is the random number of transactions used to compute the Merkle hash. The value of $RTrans$ for regular transactions is 0, and $RTrans$ for postediting transactions is computed by the $ChamCol$ function.
- $\mathcal{B}_N(PreBlock, NextBlock, ChamParam(hk, tk), RHeader, PreHash, ChamPreHash, TxRoot)$: The block $\mathcal{B}_N$ where the transaction τ is located. $PreBlock, NextBlock$ is the index of $\mathcal{B}_N$'s previous block $\mathcal{B}_{N-1}$ and next block $\mathcal{B}_{N+1}$. $ChamParam(hk, tk)$ is the parameter needed to compute the chameleon hash. $PreHash$ is the hash obtained from the block header of $\mathcal{B}_{N-1}$ after the block header hash calculation function $HeaderCal$. $ChamPreHash$ is the hash computed by the chameleon hash calculation function $ChamHashCal$ with $PreHash$, hk and $RHeader$ as parameters.
- $\{true, false\} \leftarrow ValidateBlock(\mathcal{B}_N)$: This function is used to verify whether the block is legal. The input to the function is a block, including the block header and the block data. $RTrans$ and $SigTrans$ are validated for each transaction using the Merkle hash calculation ($MerkelCal$). $RHeader$ in the block header is verified using the block header hash calculation ($HeaderCal$). If the computations all match those recorded in block $\mathcal{B}_N$, then output $true$ (otherwise, output $false$).
- $(InvBlock) \leftarrow GetInvBlocks(NextBlock)$: This function is used to obtain invalid blocks. The next block $\mathcal{B}_{N+1}$ of block $\mathcal{B}_N$ is searched by index. In addition, it is determined whether $\mathcal{B}_{N+1}$ is an invalid block according to $\mathcal{B}_{N+1}(RHeader)$. If $\mathcal{B}_{N+1}$ is an invalid block and the return value of $ValidateBlock(\mathcal{B}_{N+1})$ is $true$, then the function returns $\mathcal{B}_{N+1}$. Otherwise, it returns $null$.
- $(MBlock, VBlock) \leftarrow GetEditVerInf(\mathcal{B}_N, \tau)$: This function is used to obtain the block $MBlock$ where $ModifiedTrans$ is located and the block $VBlock$ where $VotingRes$ is located associated with the edited transaction. $MBlock$ is obtained based on $\mathcal{B}_N$ (it is the block where $MBlock$ is 1 greater than the $\mathcal{B}_N$ block number). The $ModifiedTrans$ in that block is obtained. $VotingRes$ and the block $VBlock$ in which it is located are obtained by the index of $VotingRes$ ($TxidVres \in ModifiedTrans$).

$b \leftarrow OrdTransVed(\mathcal{B}_N, \tau)$: This validates the ordinary transaction τ. The input block $\mathcal{B}_N$ is a valid block, and the transaction τ is an ordinary transaction in it. If validation is successful, $true$ is output (otherwise, $false$ is output). First, $ValidateBlock(\mathcal{B}_N)$ is computed to verify the data authenticity of τ. Then, the invalid block ($InvBlock$) of $\mathcal{B}_N$ is obtained by the function $InvBlocks$ and the index of the next block ($NextBlock \in \mathcal{B}_N$). If no $InvBlock$ is obtained, then the transaction is validated successfully. If $InvBlock$ is obtained, the transaction τ' in the invalid block that matches the transaction τ is compared. The comparison includes all relevant fields of the two transactions and the data hash of the transaction. If the comparison results all agree, the transaction is verified successfully (if the results do not agree, the verification fails).

Algorithm 1 Ordinary Transaction Validation

```
ORDTRANSVED(B_N, τ)
    if ValidateBlock(B_N) ≠ true then return false
    select NextBlock ∈ B_N
    (InvBlock) ← −GetInvBlocks(NextBlock)
    if InvBlock = null then return true
    VerT ← −InvBlock(τ')
    if VerT ≠ τ then return false
    return true
```

$\{true, false\} \leftarrow EditedTransVed(\mathcal{B}_N, \tau)$: This validates the edited transaction τ. Input block $\mathcal{B}_N$ is a valid block, and transaction τ is the edited transaction in it. If validation is successful, $true$ is output (otherwise, $false$ is output). First, the legitimacy of the current block is verified. Then, function $GetEditVerInf$ is used to obtain the blocks where $VotingRes$ and $ModifiedTrans$ are located. $VotingRes$ and $ModifiedTrans$ are verified by using the algorithm $OrdTransVed$ to ensure they have not been edited. Finally, the hash of τ is compared with the hash recorded in $VotingRes$, and the data of τ are compared with the data recorded in $ModifiedTrans$. If all the values match, validation is successful (otherwise, validation fails).

E. Discussion

There is a trapdoor management problem in redactable blockchains implemented based on the chameleon hash. Because trapdoor exposure results in any node having the ability to forge blocks, users will question the trustworthiness of the content of transactions in the blockchain. Traditional solutions aim to keep trapdoors from being exposed as well as possible. PRBFPT circumvents this problem by directly exposing trapdoors. Next, some features of public trapdoors and PRBFPT are discussed.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

Algorithm 2 Edited Transaction Validation

EDITEDTRANSVED($\mathcal{B}_N, \tau$)
  if $ValidateBlock(\mathcal{B}_N) \neq true$ **then return** $false$
  $(MBlock, VBlock) < -GetEditVerInf(\mathcal{B}_N, \tau)$
  select $ModifiedTrans \in MBlock$
  select $VotingRes \in VBlock$
  $b1 = OrdTransVed(MBlock, ModifiedTrans)$
  $b2 = OrdTransVed(VBlock, VotingRes)$
  if $b1 \neq true \| b2 \neq true$ **then return** $false$
  select $Data \in \tau$
  $THash = \mathsf{H}(Data)$
  select $NewData \in ModifiedTrans$
  select $NewDataHash \in VotingRes$
  if $Data \neq NewData \| THash \neq NewDataHash$
  then return $false$
  return $true$

1) Public trapdoor: The trapdoor secret key of PRBFPT is consistent across all nodes, and the corresponding information is publicly recorded in the block header. This means that the trapdoor of the chameleon hash is directly made public to everyone. The verification process of the chameleon hash is automated by all nodes and is no different from verifying SHA256 in the blockchain. The editing involved in PRBFPT consists of two parts: editing the content of the transaction data and modifying the block order of an invalid block when inserting the block. Later, we will show that forging data is not feasible with these two features.

Forging transaction data is not feasible because only the data in the secure block are valid in PRBFPT. We only discuss forging the data in a secure block. The method of verifying the transaction data is to first compute the chameleon hash of the transaction and then compute the Merkle hash. The Merkle hash is generally computed using SHA-256, and creating hash collisions is almost impossible. Using the chameleon hash of a transaction to create a hash collision requires modifying the random transaction validation number. With the requirement that the initial value of the random transaction validation number is 0 in all secure blocks, forging transaction data is not feasible.

Modifying the block order is not feasible. Suppose an adversary forges a secure block and modifies the block order so that the forged block is valid. The verification function of PRBFPT can easily detect this. The validity of an edit operation requires first verifying the validity of the vote result. The vote result cannot be edited, which means that its security is consistent with the security of the blockchain consensus. An adversary can only generate the voting result by a 51% attack to falsify the data. Therefore, it is not feasible to modify the block order.

In summary, PRBFPT addresses the vulnerability of block content due to disclosing trapdoors. Thus, it is able to use alternative chameleon hash algorithms, and there is no need for trapdoor management.

2) Features of PRBFPT: **The length of the edited content is not limited.** The editing process of PRBFPT adds new block data at the end of the ledger and modifies the index of the original block, changing the block order. There is no problem of operating directly on the original block data. For example, when the modified data are longer than the original data, the edit operation will overwrite the next data, causing it to be unexecutable. Obviously, in contrast to this method of editing, PRBFPT is not limited in terms of editing length.

Multiple edits can be made to the same block. The currently given algorithm illustrates only one edit implementation. The principle of multiple edits is the same as that of one edit. It is only necessary to perform another edit operation with the block of the previous edit as the object. Accordingly, the validation algorithm needs to obtain all the historical blocks of one edit operation for validation.

Auditable editing operations. Regarding the auditability of edit permission determination, PRBFPT performs edit permission determination by voting and requires that the result of the voting be recorded on the chain and can be verified by node automation. Users can retrieve the whole voting process on the chain. Second, from the perspective of the auditability of edited content, PRBFPT requires that the edited transactions be marked. Users can easily verify their validity.

Flexible deletion of raw data. PRBFPT is very flexible in this respect, depending entirely on the needs of the application. Users can choose to delete the raw data by default, not to delete it, or delete it after a limited number of blocks.

F. Security analysis

Theorem 1. *If $\mathsf{H}(\cdot)$ is a collision-resistant cryptographic hash function and the number of malicious users participating in the voting does not exceed a certain threshold, then an adversary cannot perform improper editing. In other words, the redaction scheme is secure.*

Proof. Clearly, this process is consistent with the security of voting using smart contracts if subsequent modification behavior is not considered, relying on the fact that the number of malicious voting participants is small and does not exceed the threshold for security. Since the trapdoor of $\mathcal{CH}$ is public, any user with doubts about the original transaction τ can propose an alternative transaction τ' and easily find r' satisfying $\mathcal{CH}(\mathsf{H}(\tau.Data), r) = \mathcal{CH}(\mathsf{H}(\tau'.Data), r')$. The voting participant then considers the plausibility of the new transaction τ' replacing the old transaction τ. An adversary can propose $\hat{\tau}$ that satisfies $\mathcal{CH}(\mathsf{H}(\hat{\tau}.Data), r) = \mathcal{CH}(\mathsf{H}(\tau.Data), r)$ to challenge the reasonableness of the modification operation or $\mathcal{CH}(\mathsf{H}(\hat{\tau}.Data), r') = \mathcal{CH}(\mathsf{H}(\tau'.Data), r')$ to tamper with the modified new transaction. Since the adversary did not initiate the voting contract, the voting result and the newly generated block on the chain will not be changed in any way, and r and r' cannot be modified. Therefore, the attack can only succeed if the adversary proposes a transaction $\hat{\tau}$ that satisfies $\mathsf{H}(\hat{\tau}.Data) = \mathsf{H}(\tau.Data)$ or $\mathsf{H}(\hat{\tau}.Data) = \mathsf{H}(\tau'.Data)$. Then, a successful adversary can break the collision resistance of $\mathsf{H}(\cdot)$. If the adversary is able to manipulate the consensus of the blockchain to control the voting result, it will lead to a failure of the security of the whole blockchain. An editable blockchain with a secure voting process is a secure editing method. □

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

Furthermore, from this, we can deduce that trapdoors in the chameleon hash are not directly related to the security of blockchain editing operations and that trapdoor disclosure does not affect the security of the data. This enables our scheme to apply various chameleon hash algorithms with different characteristics.

Theorem 2. *If the blockchain is consistent before editing, then the editing operation will not cause blockchain consistency to fail; that is, it will not cause a fork to arise.*

Proof. Modules such as data repair that need to be added to the blockchain client are based entirely on functions that come with the nodes (participants) themselves, so there is no need to consider the problem that each node's block number may be different during synchronization. Therefore, only the case in which all nodes have exactly the same block data is discussed. The process of the modification scheme is as follows: (1) submitting an exception in the blockchain; (2) voting using the blockchain; (3) broadcasting a request for modifying the transaction in the blockchain based on the voting result; and (4) each node modifying the transaction based on the modification request by utilizing the modification module embedded in the node. (4) is automatically and consistently executed by all nodes only after the transactions of (1), (2), and (3) are verified on the blockchain. It can be seen that these operations are recorded and executed on the blockchain, which means that they are synchronized by all nodes; otherwise, a fork will occur. Forking is something that needs to be addressed and avoided in the original blockchain scheme, so it is not within the scope of this scheme. Therefore, if the blockchain is consistent before the modification, then performing the modification will not cause the blockchain consistency to fail. □

In addition, this scheme has other security properties.

Attack to forge the ledger. If the adversary bypasses the voting operation in the modification process, it is not possible to implement the modification by directly placing a modified block on the blockchain. The validation process of the modification operation first needs to verify the validity of the voting result using the designed verification algorithms 1 and 2. The block involved in the voting and the voting result cannot be modified by default, and the modification operation does not support modification of this part. This means that an adversary cannot bypass the voting operation to initiate an attack, and the difficulty of faking the voting process is the same as the difficulty of breaking the underlying consensus security of the blockchain.

Replay attack. If bypassing the vote is not possible, then an adversary may replay the results of the vote to modify the blockchain multiple times, but again, this is not a successful attack. PRBFPT requires that the vote for modification be accompanied by a hash of the modified data. That is, the adversary succeeds if and only if the hash of the data is the same when edited multiple times, which violates the collision resistance of the hash.

IV. Implementation

A. Modification of Ethereum

Chameleon hash selection: We choose to implement the chameleon hash calculation based on the simplest primitive algorithm [6]. Since trapdoors can be made public, they can be used to accelerate hash operations.

- $(g, x, y, p, q) \leftarrow ChamPre(\kappa)$: The secret key generation algorithm takes a security parameter κ as input and outputs the public key $g, p = kq + 1$, a trapdoor secret key x, and a computationally convenient $y (y = x^{-1} \bmod q)$. All Ethernet nodes store the parameters g, p, x, y, q needed for consistent chameleon hash computations.
- $h \leftarrow ChamHash(g, x, m, r)$: The chameleon hash computation algorithm takes a chameleon hash public key g, a trapdoor secret key x, a plaintext string m, and a verified random number R as input and outputs a chameleon hash value h. m is actually the hash value of a transaction, i.e., $m = \mathsf{H}(\tau.Data)$. The specific formula is $g^{(PreHash + x \cdot r \bmod q)} \bmod p$.
- $\{0, 1\} = ChamVer(g, x, (m, r), h)$: The chameleon hash verification algorithm takes a chameleon hash public key g, a plaintext string m, a verified random number r and a chameleon hash value h as input. $ChamHash(g, x, (m, r))$ is computed to obtain h', and it is compared with h. If the hashes h and h' are equal, then it will output 1 (otherwise, it will output 0).
- $r' \leftarrow ChamCol(x, m', (m, r))$: The chameleon hash collision generation algorithm takes a chameleon hash public key hk, a plaintext string m, a verified random number r and a new plaintext string m' as input and outputs a verified random number r' that matches the plaintext string m'; this verified random number r' will satisfy $ChamVer(hk, (m, r), h) = ChamVer(hk, (m', r'), h) = 1$. If the plaintext with the verified random number (m, r) is invalid, then $null$ will be output. The specific formula is $r' \equiv (m - m') \cdot y - r \bmod q$.

Additions to the Ethereum block header field.: The relevant index and random number are added to the block header of the Ethernet block, and the relevant functions for serializing the block are rewritten. The initial value of the random number is set to 0 in the function that generates the block, and if this initial value is not 0, the block is invalid.

Additions to the Ethereum transaction field.: A random number and vote verification address are added to the transaction field in Ethereum. When a voting verification address exists in the transaction field, it is proven to be an edited transaction.

Additions to the block editing module and detection module.: The block editing module and the module for detecting modified transactions are added when the Ethereum blocks are uploaded to the chain, which implements the one-time edit and delete function in the design and directly deletes the original data by default. The performance loss of adding the module will be explained later.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

B. Contract design and implementation

A contract mig is used to implement data uploading. The user calls mig to upload data. When a user uploads data for the first time, mig records the user's address. Each time a user invokes this contract, that user's activity and the total activity are increased by 1.

The initiator calls the repair initiation function in mig with the input parameter being the hash of the repair command. A structure is generated by mig to record this behavior, and votes for addresses recorded in the contract are recorded by this structure. We calculate cumulative activity in descending order of user activity ranking. Highly active users whose cumulative activity exceeds half of the total activity are selected for voting. Only if more than half of the addresses vote will the editorial proposal be approved.

The initiator obtains the voting result through the response function and initiates an ordinary transfer transaction based on the result. The information accompanying this transaction is the repair command constructed by the initiator, the hash of the repair-initiated transaction, and the hash of the transaction in response to the result.

C. Functional test

To verify that PRBFPT is feasible, some code modules are added to some functions of Ethereum (such as those related to data upload) to implement the one-time editing scheme in the design, and a functional test is conducted.

Test process: Common transactions are uploaded through the contract. Voting is conducted through the contract before each edit. We choose to test the overall additional increase in performance metrics for the functions that are modified during the modify and delete operations and the validation of each transaction. The degree of difference of the modified system relative to the existing Ethereum system is evaluated. Finally, we compare the gas loss of sending a normal Ethereum transaction [38] [39] [40] with a special transaction that initiates an edit operation as necessary.

Test sample: The number of transactions in a block is experimentally set to have a maximum of 50 transactions in a block. The test condition uses a block with only one special transaction in a repair or delete operation, and together with the verification module, the time of the operation on the underlying ledger is recorded. All results are obtained by averaging 1000 tests.

Experimental conditions: The machine used for the experiments is an Ubuntu 18.04 virtual machine. The software platform is VMware Workstation Pro 16.1.2, and the virtual machine has 8G of RAM. The host configuration is an Intel(R) Core(TM) i5-10400 CPU with 16G of RAM.

TABLE III: PRBFPT's validation module and 2 operations

Type	Time
Validate	488.754us
Modify	613.308us
Delete	609.197us

TABLE IV: PRBFPT's special transactions and ordinary transactions in terms of gas

Function type	Original	Special
Gas	21000	23640
Gas Price (/Wei)	1000000000	1000000000

Analysis of the experimental results: From the performance loss of the native functions of Ether and the functions of PRBFPT, the extra overhead is at the microsecond level and will only reach the millisecond level at most, which is completely negligible compared to the second-level confirmation time of Ethereum. In terms of the consumption of gas for both transactions, the gas needed for PRBFPT transactions is not much larger than the amount of gas for ordinary transactions in Ethereum, and it is feasible on the public chain.

V. Conclusion

We have described our proposed practical redactable blockchain scheme (PRBFPT). It consists of two parts, namely, voting to identify the offending content and editing block additions. PRBFPT can be implemented using any chameleon-hash algorithm, and the trapdoor of the algorithm is public knowledge. There is flexibility in choosing whether to delete the original block data associated with the edit operation, and the length of the edited data is not limited by the length of the original block data. We also presented our proposed contract-based locked voting scheme, and PRBFPT is not limited to this voting scheme. The proposed contract-based locked voting scheme can be incorporated into any voting scheme and adapted to different blockchains, as long as the final result of the voting scheme can be automatically validated by the nodes. The findings from the evaluation of our PRBFPT implementation on Ethereum demonstrated that the overhead of the additional modules will be at most at the millisecond level, which is negligible in practice.

References

[1] J. Golosova and A. Romanovs, "The advantages and disadvantages of the blockchain technology," in *2018 IEEE 6th workshop on advances in information, electronic and electrical engineering (AIEEE)*. IEEE, 2018, pp. 1–6.

[2] X. Fu, H. Wang, and P. Shi, "A survey of blockchain consensus algorithms: mechanism, design and applications," *Science China Information Sciences*, vol. 64, no. 2, pp. 1–15, 2021.

[3] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, D. Müllmann, O. Hohlfeld, and K. Wehrle, "A quantitative analysis of the impact of arbitrary blockchain content on bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 420–438.

[4] C. Hopkins, "If you own bitcoin, you also own links to child porn," https://www.dailydot.com/debug/bitcoin-child-porn-transaction-code/.

[5] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2017, pp. 111–126.

[6] H. R. Krawczyk, "T: Chameleon signatures," in *Proceedings of NDSS*, vol. 2000, 2000, pp. 143–154.

[7] G. Ateniese and B. de Medeiros, "Identity-based chameleon hash and applications," in *International Conference on Financial Cryptography*. Springer, 2004, pp. 164–180.

[8] X. Chen, F. Zhang, and K. Kim, "Chameleon hashing without key exposure," in *International Conference on Information Security*. Springer, 2004, pp. 87–98.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2024.3349855

[9] G. Ateniese, D. H. Chou, B. De Medeiros, and G. Tsudik, "Sanitizable signatures," in *European Symposium on Research in Computer Security*. Springer, 2005, pp. 159–177.

[10] W. Gao, F. Li, and X. Wang, "Chameleon hash without key exposure based on schnorr signature," *Computer Standards & Interfaces*, vol. 31, no. 2, pp. 282–285, 2009.

[11] F. Bao, R. H. Deng, X. Ding, J. Lai, and Y. Zhao, "Hierarchical identity-based chameleon hash and its applications," in *International Conference on Applied Cryptography and Network Security*. Springer, 2011, pp. 201–219.

[12] X. Chen, F. Zhang, W. Susilo, H. Tian, J. Li, and K. Kim, "Identity-based chameleon hash scheme without key exposure," in *Australasian Conference on Information Security and Privacy*. Springer, 2010, pp. 200–215.

[13] M. V. R. Kumar and N. Bhalaji, "Blockchain based chameleon hashing technique for privacy preservation in e-governance system," *Wirel. Pers. Commun.*, vol. 117, no. 2, pp. 987–1006, 2021. [Online]. Available: https://doi.org/10.1007/s11277-020-07907-w

[14] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006, pp. 89–98.

[15] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. IEEE, 2007, pp. 321–334.

[16] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 406, 2019.

[17] P. Duan, J. Wang, Y. Zhang, Z. Ma, and S. Luo, "Policy-based chameleon hash with black-box traceability for redactable blockchain in iot," *Electronics*, vol. 12, no. 7, 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/7/1646

[18] S. Xu, J. Ning, J. Ma, G. Xu, J. Yuan, and R. H. Deng, "Revocable policy-based chameleon hash," in *Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, E. Bertino, H. Shulman, and M. Waidner, Eds., vol. 12972. Springer, 2021, pp. 327–347. [Online]. Available: https://doi.org/10.1007/978-3-030-88418-5_16

[19] M. Jia, J. Chen, K. He, R. Du, L. Zheng, M. Lai, D. Wang, and F. Liu, "Redactable blockchain from decentralized chameleon hash functions," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 2771–2783, 2022. [Online]. Available: https://doi.org/10.1109/TIFS.2022.3192716

[20] Y. Tian, N. Li, Y. Li, P. Szalachowski, and J. Zhou, "Policy-based chameleon hash for blockchain rewriting with black-box accountability," in *Annual Computer Security Applications Conference*, 2020, pp. 813–828.

[21] M. Khalili, M. Dakhilalian, and W. Susilo, "Efficient chameleon hash functions in the enhanced collision resistant model," *Information Sciences*, vol. 510, pp. 155–164, 2020.

[22] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 321–340.

[23] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *Annual International Cryptology Conference*. Springer, 2017, pp. 581–612.

[24] C. Wu, L. Ke, and Y. Du, "Quantum resistant key-exposure free chameleon hash and applications in redactable blockchain," *Information Sciences*, vol. 548, pp. 438–449, 2021.

[25] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Annual Cryptology Conference*. Springer, 2013, pp. 40–56.

[26] Y. Li and S. Liu, "Tagged chameleon hash from lattice and application to redactable blockchain," *IACR Cryptol. ePrint Arch.*, p. 774, 2023. [Online]. Available: https://eprint.iacr.org/2023/774

[27] X.-Y. Li, J. Xu, L.-Y. Yin, Y. Lu, Q. Tang, and Z.-F. Zhang, "Escaping from consensus: Instantly redactable blockchain protocols in permissionless setting," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–20, 2022.

[28] G. Asharov and Y. Lindell, "A full proof of the bgw protocol for perfectly-secure multiparty computation," 2011.

[29] A. Wigderson, M. Or, and S. Goldwasser, "Completeness theorems for noncryptographic fault-tolerant distributed computations," in *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, 1988, pp. 1–10.

[30] Z. Chen, Y. Tian, and C. Peng, "An incentive-compatible rational secret sharing scheme using blockchain and smart contract," *Science China Information Sciences*, vol. 64, no. 10, pp. 1–21, 2021.

[31] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[32] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[33] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable blockchain in the permissionless setting," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 124–138.

[34] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[35] M. Alharby and A. Van Moorsel, "Blockchain-based smart contracts: A systematic mapping study," *arXiv preprint arXiv:1710.06372*, 2017.

[36] J. Neylon, "How many tweets does it take to trend?" https://jungle.marketing/news/how-many-tweets-does-it-take-to-trend/.

[37] M. F. Service, "The mathematical formula for how celebrity gossip spreads on the internet," https://www.dailymail.co.uk/sciencetech/article-1262611/The-mathematical-formula-gossip-spreads-internet.html.

[38] J. Barragan, "What is ethereum transaction gas limit?" https://www.blocknative.com/blog/ethereum-transaction-gas-limit.

[39] M. Garreau, "Ethereum 101: How are transactions included in a block?" https://medium.com/ethereum-grid/ethereum-101-how-are-transactions-included-in-a-block-9ae5f491853f.

[40] J. Hendy, "How long does it take to transfer ethereum (eth)," https://www.hedgewithcrypto.com/how-long-transfer-ethereum/.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/