SEREDACT: Secure and Efficient Redactable Blockchain with Verifiable Modification

Zhuo Xu*, Xinyi Luo*, Kaiping Xue*§, David Wei†, Ruidong Li‡
* School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China
† Department of Computer and Information Science, Fordham University, Bronx, NY 10458, USA
‡ College of Science and Engineering, Kanazawa University, Kakuma-machi, Kanazawa 920-1192, Japan
§Corresponding author, kpxue@ustc.edu.cn

2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-3986-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICDCS57875.2023.00090

***Abstract*— The immutability of blockchains is an important security feature, but applications and studies have shown that it poses some problems. For instance, harmful information and vulnerable programs can be permanently stored on public blockchains such as Bitcoin and Ethereum, causing continuous damage. Therefore, researchers proposed the redactable blockchain to delete or modify those harmful data. Existing schemes usually adopt the Chameleon hash function (CHF) to keep the block hash unchanged so that other blocks remain unaffected. However, these schemes suffer from two security problems: (i) (*unknown-version*) users cannot determine whether a received block is the up-to-date version because different versions have the same hash; and (ii) (*lazy-redaction*) miners have no motivations to update historical blocks, causing continuous spreading of data which should have been discarded. To solve the problems, we propose SEREDACT, a secure and efficient redactable blockchain protocol with verifiable modification. Specifically, we design a Merkle tree-based verification mechanism with efficient dynamic updating that supports quick version checks and forcible modification updates, and further integrate it with restricted redaction policies to guarantee security. Our security and performance analyses show that SEREDACT has adequate security as a redactable blockchain protocol and retains close efficiency compared with the immutable blockchain.**

***Index Terms*—Blockchain, Redactable Blockchain, Chameleon hash, Consensus.**

I. INTRODUCTION

Blockchain has attracted extensive attention worldwide ever since it was proposed with Bitcoin [1], spawning numerous studies of blockchain protocol optimization [2]–[6] and decentralized applications, including cryptocurrencies [7]–[10], supply chains [11], energy [12], healthcare [13], PKI (public key infrastructure) systems [14], [15], etc. By adopting a hash chain data structure and decentralized consensus, once the data is recorded on-chain, all the blockchain nodes will have the same view of it, and no one could ever modify it. This immutability is such a fantastic security feature, and the blockchain is therefore widely used to establish various decentralized trusted applications. However, with its applications and further studies, some problems come up. Note that the data stored in a block is more than transaction information, and once some illegal information or buggy contracts are published on-chain, they will permanently exist and spread in the entire blockchain system, causing persistent damages [16]–[18]. A classic case of such damage is the DAO (decentralized autonomous organization) attack [19]. One hacker spotted a flaw in the DAO contract code and managed to steal 3.6 million Ether into a personal account. Since there is no way to modify the published smart contract and completed transactions, Ethereum finally forked to stop the DAO attack and retrieve the loss.

But we cannot always solve problems by forking because it is too expensive, and if a modification demand is discovered much later, it will be too late to apply a hard fork. Therefore, researchers have been trying to find methods to modify the blockchain without considerably breaching its security, and the concept of redaction blockchain subsequently came up. Generally speaking, to modify the blockchain, we should solve two fundamental issues: (i) who can modify which part of the data by which method; and (ii) how to reach a consensus on a redaction and how to ensure global version update after a valid redaction to maintain data consistency. For clearer descriptions, we separately name the two issues as the redaction policy issue and the redaction consistency issue. Currently, the former has been widely studied, and many secure and flexible redaction policies have been proposed; but the latter one has not been effectively solved.

In terms of the redaction policy issue, existing schemes fall into two categories, Chameleon hash function(CHF)-based schemes and voting-based schemes. The CHF-based scheme was first proposed by Ateniese *et al.* in [20] whose main idea is to utilize the Chameleon hash function (CHF) to modify a block without changing its hash value. CHF is a kind of trapdoor function that one can keep the hash unchanged if and only if he/she owns a trapdoor. With this feature, we can keep up the security by strictly restricting the permission to the trapdoor. Therefore, researchers have put a lot of efforts and proposed various trapdoor permission policies, including transaction-level modification [21] and k-time modification [22]. The voting-based protocol is proposed by Deuber *et al.* in [23]. In Deuber's scheme, blockchain users can initiate a modification proposal as a transaction, and the following miners then vote for a redaction by including the transaction in their block. If a transaction is included by enough miners, it then becomes valid and all other blockchain nodes then modify their local block data accordingly.

In terms of the redaction consistency issue, however, the research results are not as good as redaction policy issue.

2575-8411/23/$31.00 ©2023 IEEE
DOI 10.1109/ICDCS57875.2023.00090

We found two security issues with redaction consistency, i.e., *unknown-version* and *lazy-redaction*, reflecting that most existing redaction blockchain protocols are vulnerable. *Unknown-version* problems refer to when a blockchain user cannot determine whether a received block is the up-to-date version due to the fact that different versions of one block have the same hash and are all valid to the blockchain when adopting CHF. Although Deuber's scheme publishes the redaction proposal as a transaction in some blocks, a user has to download and traverse all the following blocks to find whether there is a redaction proposal. This traversal should be executed for each user and each block, causing unacceptable costs. *Lazy-redaction* problem refers to some blockchain nodes that do not modify historical blocks to save cost, but then may accidentally retain harmful information. Since blockchain is established on a decentralized peer-to-peer network, these lazy-redaction behaviors will lead to a continuous spreading of harmful information in the blockchain network that should have been deleted or corrected.

In this paper, to tackle the *unknown-version* and *lazy-redaction* problems of redactable blockchain, we design a verifiable modification mechanism by using the Merkle hash tree (MHT) to package the up-to-date blockchain view. We further integrate it with strict redaction policies and propose the complete SEREDACT which is a secure and efficient redaction b ckchain peffort. In summary, this paper makes the following contributions:

- We discover two significant security problems, i.e., *unknown-version* and *lazy-redaction*, that widely exist in the current redaction blockchain protocols. Aiming at solving the problems, we design a modification verification mechanism that utilizes the MHT to package the up-to-date blockchain view into a short byte string called *redactRoot* and adds it to the block header. It helps users with checking block versions and forces miners to process modifications.
- The infinite growth of blockchain and extremely expensive node-adding of MHT (reconstructing the entire MHT for adding one new node) cause severe efficiency problems. To this end, we design the *KRM-restriction* and an efficient dynamic node-adding protocol for MHT to avoid the frequent reconstruction. Also, we introduce our work on integrating the proposed verification mechanism with the existing permission control redactable blockchain protocol and further propose SEREDACT as a secure and efficient redactable blockchain protocol.
- We prove that SEREDACT effectively solves the above problems of redaction blockchain and satisfies the basic blockchain properties, including chain growth, chain quality, and common prefix [4]. We also prototype SEREDACT and our experiment results show that SEREDACT brings only a small overhead compared with the original immutable blockchain protocol.

The rest of this paper is organized as follows. Section II introduces blockchain basics, Merkle hash tree, and Chameleon hash functions. Section III explains the *unknown-version* and *lazy-redaction* problems and uses a strawman solution to illustrate our main idea for solving them. Section IV introduces the full-version of SEREDACT protocol. Section V and VI provide security and performance analysis. Section VII presents the related work and Section VIII concludes the paper.

II. PRELIMINARIES

A. Blockchain Basics

We refer to the notation in [4] to describe the blockchain and make some minor modifications to better suit our scheme. A block B consists of a block header $B.h$ and a body $B.b$. $B.b$ is a list of transactions. $B.h$ is a triple of the form $B.h := \langle s, x, ctr\rangle$, where s is the hash of the previous block, and ctr is the nonce of the proof of work (PoW) consensus. (Different consensus algorithms have different verification fields, and we take PoW as an example to introduce our scheme more clearly.) A block header also includes other important information, such as version, height, timestamp, the Merkle root of transactions in $B.b$, etc. Since these specific contents are not affected in our scheme, we generally use x to denote all of them. Based on these notations, a block B is valid if its contents are valid ($\mathrm{ValidateContent}(B)$) and ctr is a correct solution for PoW consensus ($\mathrm{ValidatePoW}(B, \mathcal{C})$). Further, a blockchain, denoted by $\mathcal{C}$, is simply a chain of blocks, i.e., $\mathcal{C} := B_1 \| B_2 \| ... \| B_n$. To verify whether a block B is a valid block of $\mathcal{C}$, one should first verify whether the block itself is valid and then verify whether it is in the chain by checking the previous hash (s) block by block ($\mathrm{CheckChain}(B, \mathcal{C})$), i.e.,

$$\begin{aligned}\mathrm{ValidateBlock}(B, \mathcal{C}) := &\ \mathrm{ValidateContent}(B)\\ &\wedge \mathrm{ValidatePoW}(B, \mathcal{C})\\ &\wedge \mathrm{CheckChain}(B, \mathcal{C}).\end{aligned}$$

For clarity, we use the following notations. The rightmost block (B_n in the example) is denoted by $\mathrm{Head}(\mathcal{C}) = B_n$ since it is usually called the chain head. The length of $\mathcal{C}$ is $\mathrm{Len}(\mathcal{C}) = n$. The height of B_j is denoted by $\mathrm{Height}(B_j) = j$. $\mathcal{C}^{\lceil k}$ denotes the chain that removes $\mathcal{C}[-k{:}]$, and analogously ${}^{\lceil k}\mathcal{C}$ denotes the chain that removes $\mathcal{C}[{:}k{+}1]$; note that if $k \geq n$ then $\mathcal{C}^{\lceil k} = \varepsilon$ and ${}^{\lceil k}\mathcal{C} = \varepsilon$ (ε denotes empty set). If $\mathcal{C}$ is a prefix of $\mathcal{C}'$, we write $\mathcal{C} \prec \mathcal{C}'$.

B. Merkle Hash Tree

A MHT [24] is a hash-based binary tree used for efficient data verification and is widely adopted in blockchain systems. By constructing an MHT of a set of data objects $o_1, ..., o_n$, we can quickly detect inconsistencies between different sets and verify whether an element is in the set without knowing the entire set. We use the following notations in this paper:

- $\mathrm{MHT}(\langle o_1, ..., o_n\rangle)$. Construct a Merkle hash tree and return the entire tree. Each leaf of the MHT is the hash of each object, i.e., $leaf_i = \mathrm{H}(o_i), i \in [1, n]$, and each branch node is the hash of its two children. For simplicity, we also use $\mathrm{MHT}(root)$ to return the entire Merkle tree corresponding to the root, i.e., $root$.

- $\mathrm{Root}(MHT)$. Return the root of a Merkle hash tree.
- $\mathrm{SetLeaf}(n, o)$. Change the n-th leaf to o, and update the related path; other nodes remain unchanged.
- $\mathrm{IsLeaf}(\mathrm{H}(\mathrm{o}))$. Given a leaf node and the path from the leaf up to the root, check whether a given hash $\mathrm{H}(o)$ is a leaf of a Merkle hash tree. This has been used in simple payment verification (SPV) in Bitcoin [1].

C. Chameleon Hash Functions

The concept of Chameleon hash was proposed by Krawczyk and Rabin [25]. A Chameleon hash function is a trapdoor hash function [26] that allows one to keep the hash value unchanged even when the message changes. Without the trapdoor, it is hard to find collisions. Its details are as follows.

- $\mathrm{Gen}(1^{\alpha})$: Given the security parameter α, the key generation algorithm outputs the public key, pk, and secret key, sk, for the Chameleon hash.
- $\mathrm{CHash}(pk, m, \lambda)$: Given the public key, pk, data, m, and a random number, λ, hash algorithm outputs a hash value, hv, and the random number, ξ.
- $\mathrm{VerifyHash}(pk, m, (hv, \xi))$: Given the public key, pk, data, m, hash value, hv, and the random number, ξ, verification algorithm checks whether (hv, ξ) is a correct hash. If so, returns 1; otherwise, returns 0.
- $\mathrm{Collision}(sk, m')$: Given the private key, sk, and the new data, m', collision algorithm outputs a new random number ξ', making $\mathrm{VerifyHash}(pk, m', (hv, \xi')) = 1$.

III. Problems, Strawman Solution, and Its Limitations

In this section, we first explain the *unknown-version* and *lazy-redaction* problems of the redactable blockchain. Then, we present a strawman solution to give an overview of our main idea to solve the two problems. We mainly introduce the block structure and basic protocols and then analyze its limitations. Aiming at releasing these limitations, we will propose our complete scheme in Section IV.

A. Unknown-version and Lazy-Redaction Problems

Given that a user, Alice, modifies a block B_j when miners are mining block B_n. After successfully generating B_n, another user, Bob, requests B_j. Fig.1(a) illustrates how the two problems, i.e., the *unknown-version* and *lazy-redaction*, happen.

- *Unknown-version.* For the Chameleon hash-based protocols, the adoption of the Chameleon hash makes the block hash unchanged, and different versions of a block have the same hash. Therefore, a user cannot determine whether a block is an up-to-date version unless he/she received a newer version.
- *Lazy-redaction.* For redactable blockchain protocols, miners may not locally redact blocks on their own if redactions in the blockchain do not affect mining. Finally, nodes in the network cannot reach a consensus on the state of the redactable blockchain, causing the old version of the block to propagate in the network.

We analyze how the two problems affect the existing redactable blockchain protocols. We find that most of them cannot resist the problems and face security vulnerabilities, as TABLE I shows. To ensure coherence, we only provide our conclusion here, and a more detailed explanation can be found in section VII.

TABLE I
Resistant ability against the two problems existing in redactable blockchain protocols

Protocols	*Basic Method*[1]	*Unknown-Version*	*Lazy-Redaction*
AMVA17 [20]	CHF	✗[2]	✗
DSSS19 [21]	CHF	✗	✗
DMT19 [23]	Voting	❍	✗
XNMHD21 [22]	CHF	✗	✗
MXNHD22 [27]	CHF	✗	✗
TLLSZ20 [28]	CHF	✗	✗

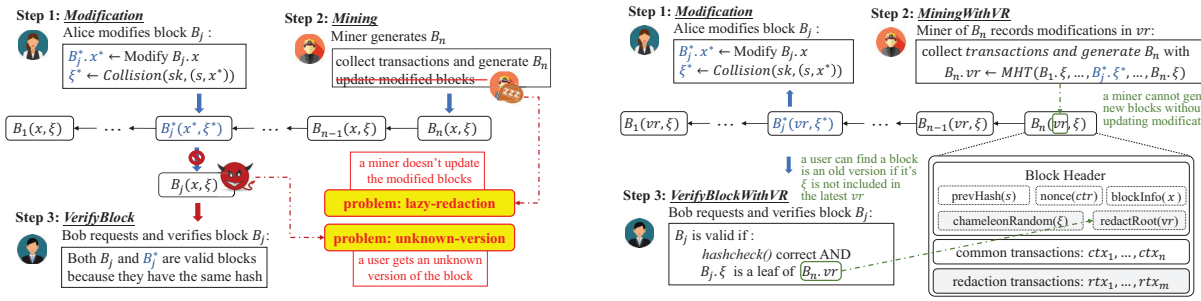
[1] ***Basic method*** refers to the underlying technology that the protocol uses to realize redaction. As we explain in section I, there are currently two types of redactable blockchain, i.e., Chameleon hash function (CHF)-based protocols and voting-based protocols.

[2] ✗ means the protocol cannot resist the related security problem, ✔ means the protocol can resist the related security problem, ❍ means the protocol may address the problem in a horribly inefficient way.

B. Strawman Solution

Our main idea is to record the latest version of all of the blocks (the state of the redactable blockchain) in the latest block. Then, users can verify a single block by it. Meanwhile, miners are forced to redact blocks locally to maintain the redaction information on the chain. Fig. 1 shows the block structure of the proposed redactable blockchain. Compared with a classical block $B := \langle s, x, ctr \rangle$, there are three new domains, i.e., the *chameleonRandom* (ξ) and *redactRoot* (vr) in the block header, and the redaction transactions $(rtx_1, ..., rtx_m)$.

- *chameleonRandom (ξ)* is used for computing the collision of Chameleon hash. As shown in Fig. 1, when Alice tries to modify the content x of block B_j to x^*, she could compute a new chameleonRandom ξ^* through *Collision*$(sk, (s, x^*))$ (see Section II).
- *redactRoot (vr)* is a Merkle hash tree root of the up to date *chameleonRandom* of all the historical blocks, e.g., $vr \leftarrow \mathrm{Root}(\mathrm{MHT}(B_1.\xi, ..., B_n.\xi))$ if the current block is with height n. Since there is a one-to-one correspondence between ξ and the block content x, vr thereupon records the latest version of all the blocks. And as shown in Fig. 1, Bob can verify whether block B_j is the up-to-date version by checking whether $B_j.\xi$ is a leaf of $B_n.vr$. Besides, a lazy miner cannot generate a valid block because he/she must update all the valid modifications to correctly compute vr. Therefore, vr solves the *unknown-version* and *lazy-redaction* problems caused by Chameleon hash.



(a) unknown-version and lazy-redaction problems of redactable blockchain.

(b) high-level overview of the proposed scheme to solve the problems.

Fig. 1. The two problems of redactable blockchain and a strawman solution that illustrates our main idea to solve the problems.

- *Redaction transactions* $(rtx_1, ..., rtx_m)$ are the transactions used to redact a block. It should have at least two items: the height of the target block and the new *chameleonRandom* ξ after changing the block data. Besides, a redaction transaction might include other information according to the adopted redaction policy, such as information about the modifier. We omit it since this does not affect our scheme.

Now, given that Alice modifies block B_j and Bob requests it, let's take a look at the workflow when adopting the *redactRoot* for redaction verification. As Fig. 1(b) shows, there are three protocols executed by three entities. The modifier (Alice) runs the *ModifyBlock* protocol to modify block B_j. Then the blockchain miners run the *MiningWithVR* protocol to generate a new block B_n. And the verifier (Bob) has a demand to verify whether a block B_j is the up-to-date version by the *VerifyBlockWithVR* protocol.

Protocol: ModifyBlock. Alice modifies some contents of block $B_j := \langle s, x, \xi, ctr, vr\rangle \| \langle ctx\rangle \| \langle rtx\rangle$ and computes the new *chameleonRandom* to get a modified block $B_j' := \langle s, x', \xi', ctr, vr\rangle \| \langle ctx'\rangle \| \langle rtx'\rangle$.

1) *Setup.* Alice acquires the Chameleon hash private key sk of B_j. The concrete process depends on the redaction policy adopted by the blockchain system, and we will discuss it later in Section IV-D.
2) *Modify.* In general, modifying a block refers to modifying one or several transactions in the block, and the corresponding fields in the block header should also be modified. We denote the modified block data as x'.
3) *Update Random.* Compute a new *chameleonRandom* through $\text{Collision}(sk, (s, x'))$ to keep the block valid.

Protocol: MiningWithVR. Miners collect all the transactions $\langle ctx\rangle, \langle rtx\rangle$ and verify each rtx with its related candidate block.

1) *Mining.* To generate a valid block, miners collect transactions and package them into one block as the block body. Then they generate the common block header including s, x, and a randomly generated ξ.
2) *ComputeVR.* For all of the candidate blocks which conform to the predetermined policy, miners should reconstruct $redactMHT$. As Fig.1 shows, given a chain $\mathcal{C} := (B_1, B_2, ..., B_n)$ and candidate blocks conform to the predetermined policy $(B_i', ..., B_j')$, miners compute $redactMHT = \text{MHT}(B_1.\xi, ..., B_i'.\xi, ..., B_j'.\xi, ..., B_n.\xi)$ and set $vr = \text{Root}(redactMHT)$. Then they create a new block $B := \langle s, x, \xi', vr, ctr\rangle \| \langle ctx\rangle \| \langle rtx\rangle$ such that $s = H(B_n.ctr, \text{CHash}(pk, B_n.x, B_n.\xi))$.

Protocol: VerifyBlockWithVR. Bob needs data in block B_j and should check whether the received B_j is the up-to-date valid version.

1) *ValidateBlock.* Bob first checks whether B_j is valid by $\text{ValidateBlock}(B, \mathcal{C})$.
2) *CheckVR.* Suppose the current $\text{Head}(\mathcal{C})$ is B_n. Then, Bob checks whether $\text{H}(B_j.\xi)$ is a leaf of $\text{MHT}(B_n.vr)$ by $\text{IsLeaf}(\text{H}(B_j.\xi), \text{MHT}(B_n.vr))$.

C. Strawman Limitations

We can easily find that the strawman solution already solves the *unknown-version* and *lazy-redaction* problems through *VerifyBlockWithVR* and *MiningWithVR* protocols, respectively. However, it has poor utility due to the *infinite growth* and *frequent reconstruction* problems, and is also insecure due to the lack of redaction policy, as follows.

- *Infinite Growth*. In the strawman solution, the *redactRoot* is the Merkle hash root of all the previous blocks' *chameleonRandom*. This means that the Merkle hash tree will infinitely grow with the growth of blockchain, leading to an increased computational and storage overhead.
- *Frequent Reconstruction*. To compute a Merkle hash root, one should compute the hash of each pair of tree nodes from the bottom layer to the top. Therefore, in the strawman solution, each newly generated block will add a new leaf to the Merkle hash tree of *chameleonRandom*, and the miner should reconstruct the entire tree every time when a new block is generated, leading to extremely expensive computational overhead.
- *Lack of Policy*. Enabling redaction in blockchain reduces the security of blockchain to some extent. Therefore, to guarantee security while enabling redaction, a redactable blockchain protocol should provide a strict redaction policy to specify who can modify which part of a block and how to realize it. The strawman solution omits the

policy for simplicity, and we will complement it with existing work in Section IV.

The above problems make the strawman solution gradually become unacceptable and eventually crash. Besides, it is also not secure due to the lack of a redaction policy. To this end, we further design the SEREDACT protocol that solves the above limitations of the strawman solution, providing a secure and efficient redactable blockchain protocol.

IV. SEREDACT PROTOCOL

A. Overview

SEREDACT aims at solving the *unknown-version* and *lazy-redaction* problems that widely exist in current redactable blockchain protocols. In the previous section (Section III), we provide a strawman solution to illustrate our main idea for solving the problems, and also show the limitations of the strawman solution. Therefore, in this section, we propose SEREDACT that overcomes the limitations from three primary aspects: (i) restrict only the rightmost k blocks to be modifiable (named *KRM-restriction*) to limit the size of *redactMHT*, and add a *solidView* field to the block header for verifying the unmodifiable blocks (Section IV-B); (ii) design a dynamic update protocol for the *redactMHT* based on the *KRM-restriction* to decrease the overhead of generating vr from reconstructing the entire *redactMHT* for updating a few paths (Section IV-C); and (iii) integrate the proposed scheme with the existing secure redaction policy (Section IV-D) to construct a secure and efficient redactable blockchain protocol (Section IV-E). Besides, TABLE II lists the main notations and functions used in SEREDACT.

TABLE II
IMPORTANT NOTATIONS AND FUNCTIONS

Notation or Function	Meaning
k	only the k-rightmost blocks are modifiable
B_n	Block at height n
ξ	chameleon hash random number
vr	verification Merkle hash tree's root
sv	*solidView* in the block
ctx	common transactions
rtx	redaction transactions
$\mathrm{Height}(B)$	get the height of the block B
$\mathrm{SetLeaf}(leaf, index)$	set the leaf of the Merkle tree
$\mathrm{IsLeaf}(leaf, tree)$	verify if the leaf belong to Merkle tree
$\mathrm{GenRedactInfo}()$	generate the vr and sv in the new block
$\mathrm{CheckVersion}(B, \mathcal{C})$	check if the block B is the up to date version

B. KRM-Restriction and solidView

To solve the *infinite growth* problem of the *redactMHT*, we restrict only the k-rightmost blocks are modifiable (named *KRM-restriction*). As a result, the *redactMHT* only needs to contain the previous k blocks rather than the entire blockchain. However, this makes it unable to verify the more previous blocks because the corresponding verification information (i.e., the hash of ξ as a leaf) will be deleted from the *redactMHT* and thereupon deleted from the entire blockchain system. Therefore, we add a *solidView* field in the block header, which is the hash of the previous $k+1$ block's ξ. The details of the design are as follows.

1) KRM-restriction: *KRM-restriction* means that modifiers can only modify the k-rightmost blocks. Intuitively, it reduces the availability of a redactable blockchain. However, in practice, the modification demands are always generated within a limited period after the data is recorded on-chain. For example, the DAO's code was found to have vulnerabilities within 40 days after it was created, and therefore setting $k = 2^{17}$ is enough for handling DAO attacks. Moreover, illegal contents in Bitcoin can also be detected over a while [17]. Besides, most redactable blockchain-based applications such as [29]–[31] also have a limited time to modify data. In addition, we can set different k according to actual demands for different blockchain systems. In consequence, it is reasonable to restrict only the k-rightmost blocks to be modifiable, and it will, in fact, not reduce availability.

2) solidView: With *KRM-Restriction*, the *redactMHT* only includes the verification information (i.e., ξ) of the previous k blocks. For instance, suppose currently $\mathrm{Head}(\mathcal{C}) = B_n$, and we have $B_n.vr := \mathrm{Root}(MHT(B_{n-k+1}.\xi, ..., B_n.\xi))$. Then, how can we verify whether a received B_{n-k} is the up-to-date version? Although $B_{n-k}.\xi$ is included when computing $B_{n-1}.vr$, the user cannot verify B_{n-k} depending on $B_{n-1}.vr$, because miners only store $\mathrm{MHT}(B_n.vr)$ locally. To solve the issue, we add a new field, called *solidView* (sv), to the block header where $B_n.sv := \mathrm{H}(B_{n-k}.\xi)$. As a result, for $B_{n-k+1}, ..., B_n$, which are within the *KRM-restriction*, one can verify them through $B_n.vr$; and for other blocks such as $B_{n-k-m}(m \geq 0)$, one can verify it by checking whether $\mathrm{H}(B_{n-k-m}.\xi)$ equals $B_{n-m}.sv$. Since B_{n-k-m} becomes unmodifiable when successfully generating B_{n-m}, $B_{n-m}.vr$ represents the finally solidified version of B_{n-k-m}.

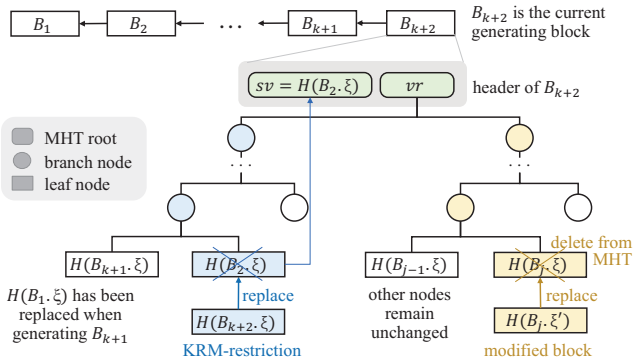


Fig. 2. Dynamic update protocol for the *redactMHT*.

C. Dynamic Update Protocol for the redactMHT

Based on KRM-restriction, we could adopt a dynamic update protocol for the *redactMHT* that decreases overhead of computing vr from reconstructing the entire *redactMHT* to

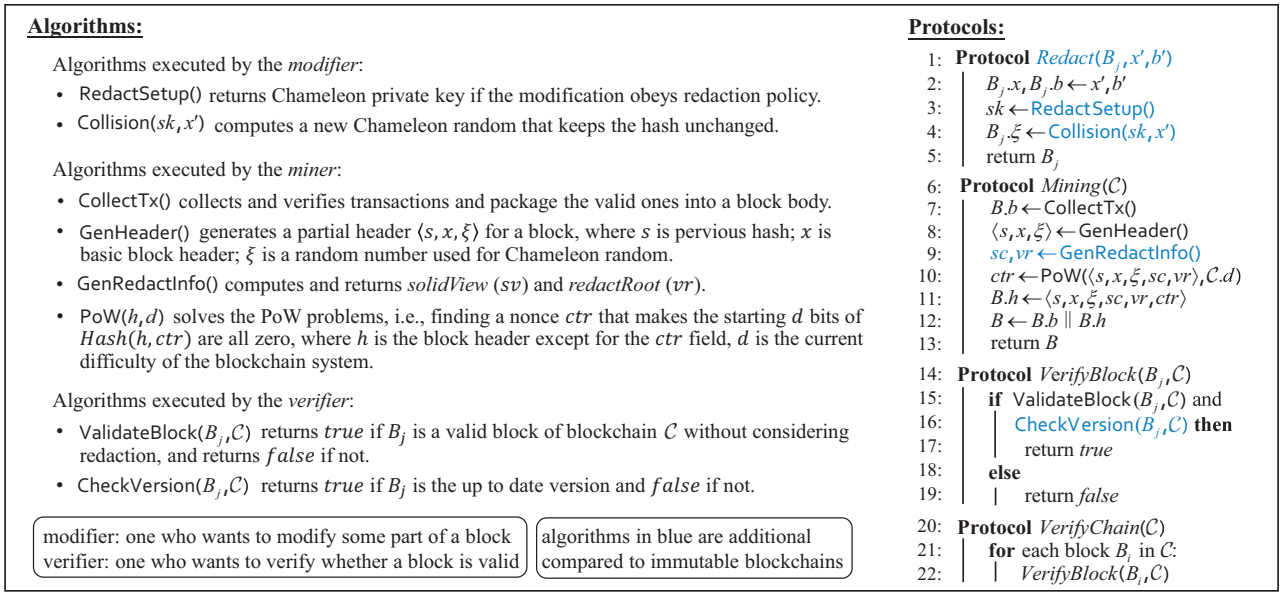


Fig. 3. Overview of algorithms (left) and protocols (right) of SEREDACT. In the protocols, those marked in blue are additional steps in SEREDACT compared to immutable blockchains such as Bitcoin.

updating a few paths. Take Fig. 2 as an example. Suppose

Algorithm 1: GenRedactInfo()

Input: A partial block $B := \langle s, x, \xi \rangle \| \langle ctx \rangle \| \langle rtx \rangle$.
Output: $\{sv, vr\}$

if $\text{Height}(B) = 1$ **then**
  $sv \leftarrow \phi$, $vr \leftarrow \text{MHT}(\text{H}(\xi), \phi, ..., \phi)$
end
else if $2 <= \text{Height}(B) <= k$ **then**
  $sv \leftarrow \phi$, $vr \leftarrow \text{SetLeaf}(n, \text{H}(\xi))$
end
else
  $n \leftarrow \text{Height}(B)$
  if *there is a* $rtx \in \langle rtx \rangle$ *that modifies* B_{n-k} **then**
    $sv \leftarrow \text{H}(rtx.\xi)$
  end
  else
    $sv \leftarrow \text{H}(B_{n-k}.\xi)$
  end
  $vr \leftarrow \text{SetLeaf}(n \mod (k), \text{H}(B.\xi))$
  for *each* $rtx \in \langle rtx \rangle$ **do**
    $vr \leftarrow \text{SetLeaf}(rtx.height \mod (k), rtx.\xi)$
  end
end
return sv, vr

that miners are generating B_{k+2} and should compute *solidView* sv and *redactRoot* vr. According to *KRM-restriction*, B_2 will become unmodifiable after B_{k+2} is added to the blockchain, and we therefore record $H(B_2.\xi)$ into B_{k+2} by setting $B_{k+2}.sv = H(B_2.\xi)$. Note that during generating B_{k+2}, B_2 is still modifiable; and if a modifier modifies B_2 to B_2' and the corresponding redaction transaction is included in B_{k+2}, the miner will set $B_{k+2}.sv = H(B_2'.\xi')$. For updating the *redactMHT*, we can easily find that $\text{H}(B_2.\xi)$ should be removed and $\text{H}(H(B_{k+2}).\xi)$ should be added as a new leaf. Therefore, intuitively, we can just replace $\text{H}(B_2.\xi)$ with $\text{H}(H(B_{k+2}).\xi)$. As a result, for each newly generated block, the overhead of computing vr is decreased from reconstructing the entire *redactMHT* to update just one path, as shown in the blue part of Fig. 2. In addition, the leaves of those newly modified blocks should also be replaced by the new version, and the corresponding paths need to be updated, as shown in the yellow part of Fig. 2. We propose Algorithm $\text{GenRedactInfo}()$, as shown in Algorithm 1, for computing sv and vr.

With sv and vr, a user can verify whether a received block B_j is the up-to-date version of blockchain $\mathcal{C}$ through $\text{CheckVersion}(B_j, \mathcal{C})$ algorithm, as shown in Algorithm 2. Note that the user is supposed to verify whether B_j is valid without considering redaction by $\text{validateBlock}(B, \mathcal{C})$ as explained in Section II-A; and if not valid, it makes no sense to verify the version. In other words, it takes two steps to verify a block in SEREDACT: (i) $\text{validateBlock}(B, \mathcal{C})$ and (ii) $\text{CheckVersion}(B_j, \mathcal{C})$.

D. Strict Redaction Policy

Chameleon hash uses a secret trapdoor (sk) to control modification permissions, i.e., only the one with sk can successfully compute the new ξ through *Collision()*. Therefore, a redactable blockchain should provide a secure redaction policy that specifies who can modify which part of a block by what method, and we denote it as *Policy(ID, Content, Verify()). ID*

Algorithm 2: $\mathrm{CheckVersion}(B, \mathcal{C})$

Input: A block $B := \langle s, x, \xi, sv, vr, ctr\rangle \| \langle ctx\rangle \| \langle rtx\rangle$.
Output: $\{true$ or $false\}$

```
j, n ← Height(B), Len(C)
if j <= n − k then
    if H(B.ξ) = B_{j+k}.sv then
        return true
    end
end
else
    if IsLeaf(B.ξ, MHT(B_n.vr)) then
        return true
    end
end
return false
```

specifies the modifier. *Content* contains the redaction and the extra information for validation. *Verify(ID, Content)* provides a method to verify whether the entity corresponding to the ID can modify the *Content*. In Fig. 3, if the modification obeys the redaction policy, i.e.,

$$\mathrm{RedactSetup}(ID, Content) := \mathrm{Verify}(ID, Content).$$

the modifier can get Chameleon secret key sk. Then, we only need to transform the existing policy into such a triplet *Policy(ID, Content, Verify())*. There are some excellent schemes that propose different policies for redactable blockchain. The policy proposed in [22] is that the trapdoor of the Chameleon hash is released, and anyone who wants to modify the chain needs to make a time-locked deposit and use the token (tk) issued by the central authority (CA) to generate a signature (σ_{tk}) when rewriting the chain. Thus, the policy can be expressed as *Policy(ID, (Tx, tk, σ_{tk}), VerifySign())*. By verifying the signature and token in the *Content*, miners can determine whether the related redaction is valid. In [21], by using Chameleon hashes with ephemeral trapdoors (CHET) [32] and attribute-based encryption (ABE) [33] , only entities that possess secret keys corresponding to attributes satisfying the access policy can find collisions for specific transaction. Therefore, this policy can be easily expressed as *Policy(ID, (Tx, Collision(rk_{ID}, Tx, pk_{Tx})), VerifyHash())*, where *VerifyHash()* verifies the correctness of the collision computed by the entity corresponding to the ID. Similar to the above two examples, we can easily transform policies in the existing works into such triples to integrate them into our scheme. Now we have a strict redaction policy in our scheme denoted as *Policy(ID, Content, Verify())*.

E. SEREDACT Protocol Outline

Based on the above designs, we propose SEREDACT which is a secure and efficient redactable blockchain protocol. Our design can be easily adopted by most of the redactable blockchain protocols that concentrate on managing redaction permission or policy. In brief, SEREDACT aims at solving the *unknown-version* and *lazy-redaction* problems which are commonly existing in redactable blockchains and have not been well solved. Fig. 3 outlines SEREDACT. There are three types of entities in SEREDACT, i.e., *modifier*, *miner*, and *verifier*. They individually execute the three protocols, *Redact*, *Mining*, and *VerifyBlock*, respectively. Compared with the immutable blockchain such as Bitcoin, *Redact* is a brand-new protocol, while *Mining* and *VerifyBlock* are basically consistent with an immutable blockchain, except for two steps: $\mathrm{GenRedactInfo}$ for *Mining* and $\mathrm{CheckVersion}$ for *VerifyBlock*. Therefore, SEREDACT can also be regarded as an extension of immutable blockchain, and it can be easily integrated into various existing blockchain protocols. In another word, SEREDACT is quite versatile.

V. SECURITY ANALYSIS

In this section, we show how SEREDACT protocol solves the *unknown-version* and *lazy-redaction* problems of a redactable blockchain. Further, we verify that SEREDACT satisfies basic security properties defined in [4] as a blockchain protocol.

A. Consistency Issue of Redactable Blockchain

As explained in Section III-A, the existing blockchain protocols suffer from the *unknown-version* and *lazy-redaction* problems, and SEREDACT aims at solving them. The following analyses show why our design works.

Theorem 1. *(Defending against Unknown-Version) Each blockchain user can determine whether a received block is the up-to-date version by VerifyBlock protocol. Besides, new miners can bootstrap the correct version of the entire blockchain by VerifyChain protocol.*

Proof. As shown in Algorithm 2, one can determine whether a received block B_j is the up-to-date version by checking sv of block B_{j+k} or vr of block B_n, according to the size relationship between j and $n-k$. The $\mathrm{CheckVersion(B, \mathcal{C})}$ would fail only when an adversary tamper with sv or vr. However, this will never happen because sv and vr are not hashed by the Chameleon hash, meaning they are unmodifiable. Similarly, by executing the *VerifyChain* protocol when bootstrapping, a new miner can ensure that he/she has acquired the up-to-date version of the entire blockchain. □

Theorem 2. *(Defending against Lazy-Redaction) A lazy miner who does not process modifications cannot continuously generate new blocks, and a malicious miner who deliberately retains the old versions cannot further spread them in the network.*

Proof. To generate a new block B_{new}, a miner should first compute the *solidView* (sv), the *redactMHT* and *redactRoot* (vr) through the $\mathrm{GenRedactInfo()}$ algorithm (see Algorithm 1). Since the leaf of *redactRoot* is the hash of the up-to-date *chameleonRandom* (ξ) of the k-rightmost blocks, the miner should collect all the modifications to update the *redactMHT*. If the miner misses some modifications, the *redactRoot* in B_{new} is different from other honest miners, and other nodes will not accept B_{new}. Since blockchain users can determine

whether a received block is the up-to-date version by *VerifyBlock* protocol, it is meaningless for a miner to retain the old versions and spread them in the network. □

B. Basic Properties of Blockchain

As defined in [4], a blockchain protocol should satisfy three basic security: chain growth, chain quality, and common prefix. However, as proved in [23], a redactable blockchain inherently does not satisfy the common prefix property; instead, a redactable blockchain is secure as long as it satisfies the editable common prefix property, and our design inherits this definition. In this section, we prove the security of SEREDACT from these aspects.

Theorem 3. *(Chain Growth [4])* SEREDACT *satisfies the chain growth property, that is, for any honest party $\mathcal{P}$ that has the chains $\mathcal{C}_1, \mathcal{C}_2$ at the onset of the two slots sl_1, sl_2, if it has $sl_2 - sl_1 \geq \tau \cdot s$, then we have $\text{len}(\mathcal{C}_2) - \text{len}(\mathcal{C}_1) \geq \tau \cdot s$, for $s \in \mathbb{N}$ and $0 < \tau \leq 1$, where τ is the speed coefficient.*

Proof. In SEREDACT, the chain growth property is still preserved since there is no operation that can alter the length of a chain. In general, if a redactable blockchain protocol Π does not have any edit operation that can alter the length of the chain, protocol Π with our verification mechanism satisfies the chain growth property. □

Theorem 4. *(Chain Quality [4])* SEREDACT *satisfies the chain quality property, that is, for any honest party $\mathcal{P}$ that has a chain $\mathcal{C}$, the ratio of honest blocks in a portion of length ℓ-blocks of $\mathcal{C}$ is at least μ, where $0 < \mu \leq 1$ is the chain quality coefficient.*

Proof. In SEREDACT, all of the redactions satisfy the strict policy. Thus, an adversary could not decrease the proportion of honest blocks, and the chain quality property is thus preserved in the redactable blockchain protocol. If a redactable blockchain protocol Π satisfies (μ, ℓ)-chain quality, then the extension of Π to SEREDACT protocol still satisfies (μ, ℓ)-chain quality. In a redactable blockchain protocol Π, an adversary could decrease the proportion of honest blocks by redacting some honest blocks. But the verification mechanism in SEREDACT does not provide a way to redact a block. Therefore, SEREDACT would not influence the quality of the chain. □

Theorem 5. *(Editable Common Prefix [23])* SEREDACT *satisfies the editable common prefix property, that is, for any pair of honest parties $\mathcal{P}_1, \mathcal{P}_2$ adopting the chains $\mathcal{C}_1, \mathcal{C}_2$ with length l_2, l_1 at the onset of the slots $sl_1 \leq sl_2$, we have one of the following:*

1) $\mathcal{C}_1^{\lceil m} \preceq \mathcal{C}_2$,
2) *for each $B_j \in \mathcal{C}_2^{\lceil (l_2 - l_1) + m}$ such that $B_j \notin \mathcal{C}_1^{\lceil m}$ and $j \leq (l_2 - k)$, it holds that $B_{j+k}.sv = B_j.\xi$,*
3) *for each $B_j \in \mathcal{C}_2^{\lceil (l_2 - l_1) + m}$ such that $B_j \notin \mathcal{C}_1^{\lceil m}$ and $(l_2 - k) \leq j$, it holds that $\text{IsLeaf}(B_j.\xi, \text{MHT}(Head(\mathcal{C}_2).vr))$ is true.*

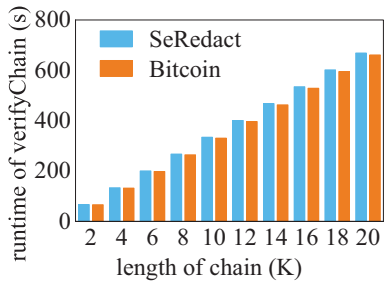
where $m \in \mathbb{N}$ denotes the common prefix parameter and k denotes the system parameter of the SEREDACT *protocol proposed in Section IV.*

Proof. For any redaction in SEREDACT, it must be confirmed by all the nodes in the network through the verification mechanism. Then for each $B_j \in \mathcal{C}_2^{\lceil (l_2 - l_1) + m}$ such that $B_j \notin \mathcal{C}_1^{\lceil m}$, it could pass the $VerifyBlock(B_j, \mathcal{C})$. In general, any redactable blockchain protocol Π with the proposed verification mechanism of SEREDACT satisfies the editable common prefix property. □

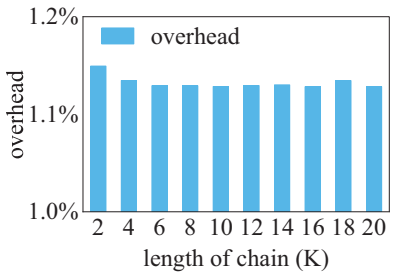
VI. PERFORMANCE ANALYSIS

A. Implementation

To demonstrate the efficiency of SEREDACT protocol, we prototype it and conduct some experiments on macOS Catalina (v10.15.7) with Intel Core i5 CPU @2 GHz and 16 GB RAM. The basic redactable blockchain protocol is based on the framework proposed by Ateniese *et al.* [20] and its algorithm to shrink the chain is removed since it is not compatible with SEREDACT. We implement this redactable Bitcoin with SEREDACT protocol on Python 3 using the python-bitcoinlib (v0.11.0). The cryptographic parts are implemented based on PyCryptodome (v3.10.1) and gmpy2 (v2.0.8), and parameters of Chameleon hash are generated by SageMath (v9.2).



(a) Comparison of the performance of immutable chain and redactable chain with verification mechanism.



(b) Validation time overhead required to validate a redactable chain with verification mechanism compared to an immutable chain.

Fig. 4. Time consumption of *VerifyChain* protocol.

B. Performance of Verification

SEREDACT mainly consists of two verification protocols: *VerifyChain* and *VerifyBlock*. Since the *VerifyBlock* protocol is

the unit process of *VerifyChain*, we mainly use the results of *VerifyChain* to show the performance as a whole. We first evaluate the runtime of *VerifyChain* protocol and make a performance comparison with immutable blockchain. We generate blockchains with total length ranging from 2000 to 20000 when setting $k = 2^{10}$. Fig. 4(a) and Fig. 4(b) show that the runtime of both SEREDACT and immutable blockchain grow linearly and SEREDACT only introduces 1% additional overhead compared to immutable blockchain. Due to KRM-restriction, i.e., the size of the Merkle tree is fixed and the time consumption from computing the *redactMHT* is fixed, this overhead keeps a limit value and even decreases when the chain length increases, as Fig. 4(b) shows.

The process of validating the chain is divided into three parts: (i) validate transactions; (ii) validate block headers; (iii) verify the version of the blocks in the chain. Compared to immutable blockchain, the extra time consumption is from computing the *chameleonRandom* (ξ) and the *redactMHT*, as shown in TABLE III.

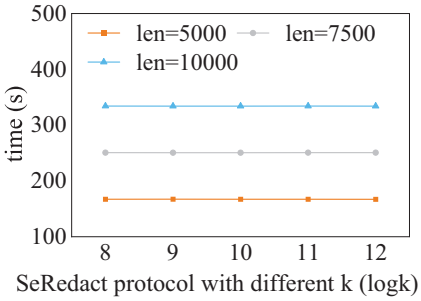
TABLE III
RUNTIME(S) OF VALIDATING THE BLOCKCHAIN OF SEREDACT AND BITCOIN

Protocols	VerifyTx	VerifyBlock	VerifyVersion	Total
SEREDACT	342.56	4.25	0.02	346.83
BITCOIN	342.56	0.54	0	343.10

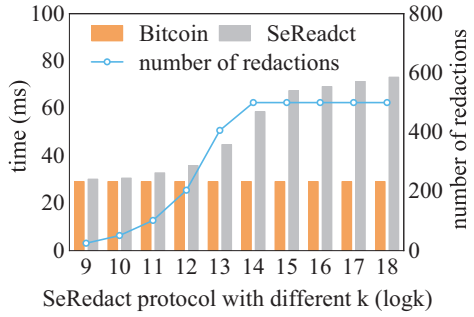
To evaluate how the parameter (k) of *KRM-restriction* affect verification, we generate SEREDACT chain consisting of 10000 blocks with k ranging from 2^8 to 2^{12}. As shown in Fig. 5(a), k has little influence on the runtime of *VerifyChain* protocol because the verification time of *redactMHT* is negligible compared with the transactions verification. Therefore, selecting a relatively large system parameter k will not cause a great impact on the *VerifyChain* protocol.

C. Performance of Mining

We evaluate the *Mining* process with different system parameter k ranging from 2^9 to 2^{18} (enough to fix DAO attack in Ethereum as introduced in Section I). As Fig. 5(b) shows, the overhead of *Mining* process compared to immutable blockchain is about tens of milliseconds which is accptable in real systems, since the mining process is mainly consumed by consensus algorithms. Compared to immutable blockchain, the extra overhead of *Mining* comes from Algorithm 1 which contains dynamic update protocol. As we mentioned in Section III-C, the strawman solution has poor utility and we design the dynamic update protocol to reduce the overhead for miners to maintain the *redactMHT*. In order to prove that our optimization is effective, we measure the runtime of the dynamic update protocol with different system parameter k ranging from 2^8 to 2^{12}. Besides, since the number of redactions ρ in the k rightmost blocks in a slot directly influences the runtime of the dynamic update protocol, revocation rate ρ should be considered in the experiment.



(a) Runtime of *VerifyChain*.



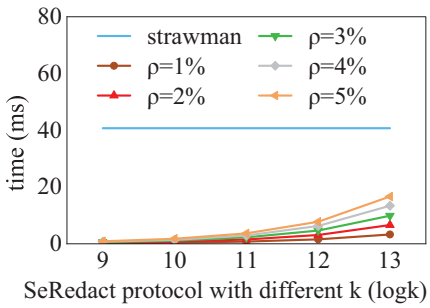
(b) Runtime of *Mining* without PoW.

Fig. 5. System performance under different system parameter k

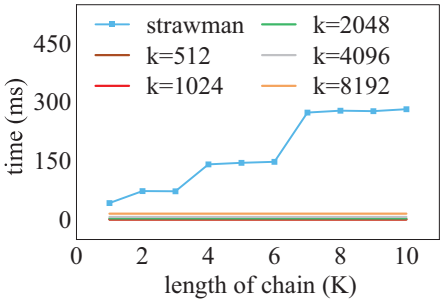
As Fig. 6(a) shows, the runtime of the dynamic update protocol increases as k and revocation rate ρ increases. Specifically, in a update process, miners should update the leaves of *redactMHT* for *KRM-restriction* and redactions in a slot, that is $1 + \rho k$ times update operation for Merkle tree. Then the algorithm complexity is $O(\rho k log(k))$, if the redaction transactions reach the max number in a block, then the complexity is $O(\log(k))$. The results in Fig. 6(a) basically satisfy the theoretical analysis. Strawman solution in Fig. 6(a) shows the time consumption of reconstructing the *redactMHT* in the slot that the length of chain is 10000 and it is greater than any case of dynamic update protocol. Fig. 6(b) shows that the time consumption of reconstructing the *redactMHT* in the strawman solution increases as the length of chain increases and it is greater than the dynamic update protocol with k ranging from 2^9 to 2^{13}, $\rho = 5\%$. As for Ethereum, the strawman solution is unacceptable since there are tens of millions of blocks. In general, the dynamic update protocol has stable performance as the length of the chain increases and only depends on the system parameter k and the revocation rate in a single slot. The complexity of the dynamic update protocol is acceptable and more practical compared to the block generation speed, which is $16s$ per block, of Ethereum.

D. Setting of Parameter k

We also consider how to set the system parameter k. The setting of parameter k depends on two aspects, i.e., the redactable period of the redactable blockchain and the impact of parameter k on system performance. The redactable



(a) Different k and ρ.



(b) Different blockchain lengths.

Fig. 6. Time consumption of updating $redactMHT$

period of the redactable blockchain means the longest time between data recording on-chain and data redaction, which is denoted as t_r. Combined with the block generation rate (g_r) of the blockchain, we can get the minimum value of k, i.e., $k = \frac{t_r}{g_r}$. For simplicity of implementation, we can set $k = 2^{\log_2 \lceil \frac{t_r}{g_r} \rceil}$. TABLE IV shows the redactable period of different blockchains with different k. The block generation rates are set as 0.1, 5, and 3 blocks per minute for Bitcoin, Ethereum, and Cardano respectively. As we mentioned before, the dynamic update protocol complexity is $O(\rho k log(k))$, if the redaction transactions reach the max number in a block, then the complexity is $O(\log(k))$. Therefore, the impact of parameter k on system performance is actually limited and we can just set $k = 2^{\log_2 \lceil \frac{t_r}{g_r} \rceil}$ for the redactable period t_r we need.

TABLE IV
REDACTABLE PERIOD(DAYS) OF DIFFERENT BLOCKCHAINS WITH DIFFERENT K

k	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
BITCOIN	56	113	227	455	910	1820
ETHEREUM	1	2	4	9	18	36
CARDANO	1	3	7	15	30	60

VII. RELATED WORK

A. Redactable Blockchain Protocols

Generally speaking, redactable blockchain protocols fall into two categories, i.e., voting-based protocols and Chameleon hash function (CHF)-based protocols. Deuber *et al.* [23] proposed a solution in permissionless setting based on voting. Chain participants propose a redacting proposal, and miners vote for the redaction. If a proposal collects enough votes, then all miners replace the original block with the candidate block. [20] proposed the first redactable blockchain by using CHF. Briefly, they use CHF to replace the regular hash function when linking the blocks in the chain. Anyone who has the trapdoor can modify the block. To make redactions controllable and fine-grained, Derler *et al.* [21] proposed a solution by using policy-based Chameleon hash based on attribute-based encryption(ABE) which makes redaction more fine-grained and it supported transaction-level rewriting. To further enhance the security of redactable blockchains, Xu *et al.* [22] proposed a solution with monetary penalty and k-time modification operation against malicious behaviors of modifiers. Tian *et al.* [28] introduced policy-based Chameleon hash with black-box accountability (PCHBA), which achieved anonymity and accountability. Since centralized authority is vulnerable to attack, Ma *et al.* [27] introduced the notion of decentralized policy-based Chameleon hash (DPCH) and proposed a decentralized rewriting mechanism without a need of trusted central authority.

B. Two Security Problems of Existing Works

As we mentioned in Section III, most existing works cannot resist the two security problems: *unknown-version* and *lazy-redaction*. For voting-based protocols, by verifying the correspondence between votes and redactions recorded in the chain, the *unknown-version* problem can be solved in a horribly inefficient way. For CHF-based protocols, *unknown-version* problem remains due to the use of CHF and the lack of redaction records in the chain. For the *lazy-redaction* problem, there is no mechanism to prevent this from happening because miners can continue mining without applying reactions in any of the existing works.

VIII. CONCLUSION

In this paper, we pointed out two security problems, *unknown-version* and *lazy-redaction*, that widely exist in current redactable blockchain protocols. We then proposed SEREDACT to tackle both problems. By utilizing the Merkle hash tree (MHT) to package the up-to-date blockchain view into the *redactMHT* and record its root into the latest block, users can check block versions with the help of *redactMHT* and it also forces miners to process all of the modifications, thereby solving both problems. Further, regarding the performance limitations caused by the blockchain's infinite growth and MHT's expensive node-adding overhead, we proposed the *KRM-restriction* and thereupon designed the dynamic update protocol for *redactMHT*. We finally integrated the proposed

scheme with existing redaction policies to enhance security for the proposed SEREDACT protocol. We proved its security, and our experiments show that the dynamic update protocol significantly reduces the overhead to maintain *redactMHT*, and SEREACT protocol brings just a little overhead compared with immutable blockchain.

ACKNOWLEDGMENT

This work is supported in part by Anhui Province Key Technologies Research & Development Program under Grant No. 2022a05020050, the National Natural Science Foundation of China under Grant No. 61972371, Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093, and JSPS KAKENHI under Grant No. JP19H04105.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/en/bitcoin-paper, 2008, Bitcoin white paper, accessed: Jul., 2021.

[2] A. Kiayia, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proceedings of the 2017 Annual International Cryptology Conference (Crypto)*. Springer, 2017, pp. 357–388.

[3] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 913–930.

[4] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Proceedings of the 2015 Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocypto)*. Springer, 2015, pp. 281–310.

[5] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *Proceedings of the 2018 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2021, pp. 1–10.

[6] J. Xu, Y. Cheng, C. Wang, and X. Jia, "Occam: A secure and adaptive scaling scheme for permissionless blockchain," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 618–628.

[7] V. Buterin, "Ethereum: A secure decentralised generalised transaction ledger," https://ethereum.org/en/whitepaper/, 2013, Ethereum white paper, accessed: Jul., 2021.

[8] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2013, pp. 397–411.

[9] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proceedings of the 2014 IEEE symposium on security and privacy (S&P)*. IEEE, 2014, pp. 459–474.

[10] Z. Guan, Z. Wan, Y. Yang, Y. Zhou, and B. Huang, "Blockmaze: An efficient privacy-preserving account-model blockchain based on zk-snarks," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[11] Y. Wang, J. H. Han, and P. Beynon-Davies, "Understanding blockchain technology for future supply chains: a systematic literature review and research agenda," *Supply Chain Management: An International Journal*, 2018.

[12] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, "Consortium blockchain for secure energy trading in industrial internet of things," *IEEE transactions on industrial informatics*, vol. 14, no. 8, pp. 3690–3700, 2017.

[13] J. Xu, K. Xue, S. Li, H. Tian, J. Hong, P. Hong, and N. Yu, "Healthchain: A blockchain-based privacy preserving scheme for large-scale health data," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8770–8781, 2019.

[14] J. Chen, S. Yao, Q. Yuan, K. He, S. Ji, and R. Du, "CertChain: Public and efficient certificate audit based on blockchain for TLS connections," in *Proceedings of the 2018 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2018, pp. 2060–2068.

[15] M. Jia, K. He, J. Chen, R. Du, W. Chen, Z. Tian, and S. Ji, "PROCESS: Privacy-preserving on-chain certificate status service," in *Proceedings of the 2021 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2021, pp. 1–10.

[16] C. Hopkins, "If you own bitcoin, you also own links to child porn," 2015. [Online]. Available: https://www.dailydot.com/business/bitcoin-child-porn-transaction-code/

[17] 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 *Proceedings of the 2018 International Conference on Financial Cryptography and Data Security (FC)*. Springer, 2018, pp. 420–438.

[18] B. Hou and F. Chen, "A study on nine years of bitcoin transactions: Understanding real-world behaviors of bitcoin miners and users," in *2020 IEEE 40th international conference on distributed computing systems (ICDCS)*. IEEE, 2020, pp. 1031–1043.

[19] D. Siegel, "Understanding the dao attack," 2016. [Online]. Available: http://www.coindesk.com/understanding-dao-hack-journalists/

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

[21] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based," in *Proceedings of the 2019 ISOC Network and Distributed System Security Symposium (NDSS)*. ISOC, 2019.

[22] S. Xu, J. Ning, J. Ma, X. Huang, and R. H. Deng, "K-time modifiable and epoch-based redactable blockchain," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4507–4520, 2021.

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

[24] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proceedings of the 1987 Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*. Springer, 1987, pp. 369–378.

[25] H. Krawczyk and T. Rabin, "Chameleon hashing and signatures," in *Proceedings of the 2000 ISOC Network and Distributed System Security Symposium (NDSS)*. ISOC, 2000.

[26] N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky, "Trapdoor hash functions and their applications," in *Annual International Cryptology Conference (CRYPTO)*. Springer, 2019, pp. 3–32.

[27] J. Ma, S. Xu, J. Ning, X. Huang, and R. H. Deng, "Redactable blockchain in decentralized setting," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1227–1242, 2022.

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

[29] J. Xu, K. Xue, H. Tian, J. Hong, D. S. Wei, and P. Hong, "An identity management and authentication scheme based on redactable blockchain for mobile networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 6, pp. 6688–6698, 2020.

[30] X. Luo, Z. Xu, K. Xue, Q. Jiang, R. Li, and D. S. Wei, "Scalacert: Scalability-oriented pki with redactable consortium blockchain enabled "on-cert" certificate revocation," in *Proceedings of the 2022 IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022.

[31] K. Huang, X. Zhang, Y. Mu, X. Wang, G. Yang, X. Du, F. Rezaeibagha, Q. Xia, and M. Guizani, "Building redactable consortium blockchain for industrial internet-of-things," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3670–3679, 2019.

[32] J. Camenisch, D. Derler, S. Krenn, H. C. Pöhls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors," in *Proceedings of the 2017 IACR International Workshop on Public Key Cryptography*. Springer, 2017, pp. 152–182.

[33] A. Lewko and B. Waters, "Decentralizing attribute-based encryption," in *Proceedings of the 2011 Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*. Springer, 2011, pp. 568–588.