entropy

Article

AeRChain: An Anonymous and Efficient Redactable Blockchain Scheme Based on Proof-of-Work

Bin Luo [1,2] and Changlin Yang [3,*]

[1] College of Information Science and Technology, Jinan University, Guangzhou 510632, China
[2] Pazhou Laboratory, Guangzhou 510330, China
[3] School of Software Engineering, Sun Yat-sen University, Zhuhai 519000, China
* Correspondence: yangchlin6@mail.sysu.edu.cn

Abstract: Redactable Blockchain aims to ensure the immutability of the data of most applications and provide authorized mutability for some specific applications, such as for removing illegal content from blockchains. However, the existing Redactable Blockchains lack redacting efficiency and protection of the identity information of voters participating in the redacting consensus. To fill this gap, this paper presents an anonymous and efficient redactable blockchain scheme based on Proof-of-Work (PoW) in the permissionless setting, called "AeRChain". Specifically, the paper first presents an improved Back's Linkable Spontaneous Anonymous Group (bLSAG) signatures scheme and uses the improved scheme to hide the identity of blockchain voters. Then, in order to accelerate the achievement of redacting consensus, it introduces a moderate puzzle with variable target values for selecting voters and a voting weight function for assigning different weights to puzzles with different target values. The experimental results show that the present scheme can achieve efficient anonymous redacting consensus with low overhead and reduce communication traffic.

Keywords: Redactable Blockchain; privacy protection; linkable ring signature; Proof-of-Work

check for updates

Citation: Luo, B.; Yang, C. AeRChain: An Anonymous and Efficient Redactable Blockchain Scheme Based on Proof-of-Work. *Entropy* **2023**, *25*, 270. https://doi.org/10.3390/e25020270

Academic Editors: Ke Xiong, Junbin Fang, Yunjiang Wang and Fan Cheng

Received: 16 December 2022
Revised: 11 January 2023
Accepted: 16 January 2023
Published: 1 February 2023

1. Introduction

Blockchain can be regarded as a decentralized database and a distributed ledger technology. Compared with traditional databases, blockchain has the important properties of decentralization and immutability. Decentralization effectively solves the problem of a single point of failure, and immutability effectively ensures the integrity of data. These superior properties enable the blockchain to build a trust bridge for nodes in an untrusted environment and ensure the authenticity and reliability of data. Therefore, industry and commerce are considering the use of blockchain technology as a means of solving many practical problems, such as in healthcare [1] and e-government [2].

However, many cases show that immutability limits the development of blockchain. Firstly, some reports mention the existence of illegal data in the blockchain, especially in bitcoin [3]. Secondly, the data stored in the blockchain is increasing, which means that the communication load in the network is constantly increasing. Thirdly, institutions using blockchain technology to provide services sometimes need to correct user data. Because the blockchain cannot be redacted, it becomes very difficult to change wrong data or delete illegal data. Finally, the EU's General Data Protection Regulation (GDPR) and other regulations require citizens to be given the "Right to be Forgotten" [4], which cannot be met by the current blockchain. Therefore, there is an urgent need for a safe and effective method to redact the data in the blockchain, and Redactable Blockchain is the general term for such methods.

Redactable Blockchain refers to a type of blockchain that allows users to redact the data on the chain in a special scenario. Its redacting operation is mainly aimed at the immutability of the blockchain, that is, to achieve the deletion, modification, and insertion of data on the chain without destroying other properties of the blockchain.

1.1. Related Work

In recent years, some scholars have begun to study blockchain redacting functions and put forward some schemes. These schemes can be roughly divided into Chameleon Hash Function (CHF)-based and voting-based redacting consensus methods.

CHF is a one-way hash function with a trapdoor, which can be used to change the input without changing the output of the function. The first redactable blockchain scheme was proposed by Ateniese et al. [5] in 2017. This scheme replaces the original hash function of the blockchain with CHF, which can modify the blockchain data while keeping the hash value unchanged. However, the scheme uses some complex cryptographic technologies which are not suitable for permissionless blockchains. Later, Derler et al. [6] proposed the concept of policy-based chameleon hash. Although the transaction owner can set a policy for redacting the transaction, and only users whose attributes satisfy the policy can redact the transaction, the scheme does not have a mechanism to revoke the trapdoor, so once the user obtains the trapdoor, he may redact data multiple times. Recently, Xu et al. [7] proposed a redactable blockchain scheme with a money punishment mechanism. There is an authority CA in this scheme, which can specify the maximum number of times that redactors can modify the data. Each redactor needs to pay a deposit, and if he acts maliciously, the CA will deduct the deposit as punishment.

In the voting-based method, when the number of votes for the data to be redacted reaches a certain threshold, it can be regarded as having reached a redacting consensus. In 2019, Deuber et al. [8] proposed a redactable blockchain scheme based on this mechanism, which adds an "old state" in the block to maintain the integrity of the hash chain. However, it took too long to reach a redacting consensus in this scheme. To solve this problem, Li et al. [9] proposed an instantly redactable blockchain scheme. In this scheme, a difficult virtual problem is used to run for voters. However, the scheme does not hide the identity of voters, and there is no corresponding voting incentive mechanism. In addition, Thyagarajan et al. [10] proposed a general redactable protocol *Reparo*, which can repair vulnerable smart contracts. However, the protocol may need to cascadingly modify the affected subsequent transactions when necessary.

In order to protect the privacy of users in the redacting process, Cai et al. [11] proposed a removable blockchain that allows users to hide transaction contents and addresses, but this scheme only realizes the deletion operation. Later, Ren et al. [12] proposed a redactable blockchain to ensure the identity privacy of users, but only the transaction sender has the right to modify the transaction. In the same year, Panwar et al. [13] proposed a framework for rewriting blockchain content using CHF and dynamic group signature technology; however, there is a central entity in the scheme.

1.2. Our Contributions

Compared with the CHF-based method, the voting-based method can avoid the trapdoor management problem and is more suitable for permissionless settings with a wide range of applications. However, the above privacy protection schemes still have problems such as third-party intervention, centralized redacting rights, and low efficiency in reaching a redacting consensus. Therefore, we have designed an anonymous and efficient redactable blockchain scheme named "AeRChain", which mainly focuses on how to enable users to participate in the whole redacting consensus process anonymously and compliantly without third-party intervention, and relatively quickly reach redacting consensus in the permissionless setting based on Proof-of-Work (PoW) consensus. The main contributions of this paper are summarized as follows:

- We improve Back's Linkable Spontaneous Anonymous Group signatures (bLSAG) scheme, introduce a moderate puzzle with variable target value as a method to select voters participating in redacting consensus, and add a voting weight function to associate the weight of votes with variable target values.
- We propose an anonymous and efficient redactable blockchain scheme called AeRChain, in which the improved bLSAG scheme is used to hide the identity of voters

participating in redacting consensus, and the moderate puzzle and voting weight function are used to accelerate the achievement of redacting consensus.

- We carry out experiments on the proposed scheme, and the results show that our scheme can achieve efficient and anonymous redacting consensus with acceptable overhead and can reduce communication traffic when the system sets reasonable target values and number of ring members.

The comparison between our scheme and related schemes is shown in Table 1, where public verifiability means that voting results and redacting proofs can be verified by any user.

Table 1. Comparison of our scheme with related works.

Scheme	Permissionless	Public Verifiability	Redaction Time	Anonymity Without Third Party
Deuber [8]	●	●	513 slots	○
Li [9]	●	●	20 slots	○
Panwar [13]	○	○	—	○
ours	●	●	$\leq$ 20 slots	●

The rest of this paper is organized as follows: In Section 2, we introduce some basics related to blockchain and cryptography. In Section 3, we show the specific structure of the improved bLSAG scheme and voting weight function, and conduct a security analysis of the improved bLSAG scheme. In Section 4, we first give the system model and threat model of the AeRChain scheme, and then describe the AeRChain scheme in detail. In Section 5, we analyze the security and experimental results of the AeRChain scheme. Finally, Section 6 concludes this paper.

2. Background

2.1. Blockchain

In this subsection, we refer to the abstract way of describing blockchain in [14,15], review the basic definition of blockchain, and use this abstract way to explain our scheme in the following article. Assume that the blockchain protocol is executed in the discrete time unit *slot*, and that $\mathcal{H}$ and $\mathcal{G}$ are hash functions in cryptography. In traditional blockchain, a block is of the form $\mathfrak{B}_i := (sl_i, ne_i, ph_i, \mathcal{G}(tx_i), tx_i)$, where sl_i is the time unit, ne_i is the solution to the underlying consensus algorithm, ph_i is the hash value of the previous block, tx_i is the transaction contained in block $\mathfrak{B}_i$, and $\mathcal{G}(tx_i)$ is the root of the Merkle tree composed of transactions tx_i. The Merkle tree is a binary tree obtained by pairwise hashing of transactions. Treat the hash value of each transaction as the leaf node of the tree, and combine two adjacent nodes into a new node from the bottom up until a unique hash value is formed at the end, which is the Merkle root. Blockchain $\mathfrak{C}$ is composed of a string of the above blocks, i.e., $\mathfrak{C} := (\mathfrak{B}_1, \mathfrak{B}_2, \cdots, \mathfrak{B}_l)$, where $\mathfrak{B}_1$ is the genesis block, $\mathfrak{B}_l$ is the head of the chain, and l is the length of the chain. Blockchain mainly uses a consensus algorithm to run for block producers, for example, the PoW algorithm used in Bitcoin. In short, the algorithm takes new block data and a given value T as inputs to find a random value ne so that $\mathcal{H}(sl, ne, ph, \mathcal{G}(tx)) < T$.

2.2. Linkable Ring Signature

Generally speaking, a linkable ring signature scheme is mainly composed of five algorithms: initialization algorithm *Setup*, key pairs generation algorithm *KeyGen*, signature algorithm *Sign*, signature verification algorithm *Verify*, and link algorithm *Link*.

- $Setup(1^\lambda) \rightarrow pp$. The algorithm takes the security parameter 1^λ as the input and outputs the common parameter pp.

- $KeyGen(pp) \rightarrow (sk, pk)$. The algorithm takes the public parameter *pp* as the input and outputs a pair of public and secret keys (*sk,pk*).
- $Sign(R, sk, m) \rightarrow \sigma$. The algorithm takes ring *R*, signer's secret key *sk*, and message *m* to be signed as inputs and outputs a signature σ.
- $Verify(\sigma, R, m) \rightarrow True/False$. The algorithm takes ring *R* and signed message *m* as inputs. If the signature is legal, it outputs True, otherwise it outputs False.
- $Link((\sigma_1, R_1, m_1), (\sigma_2, R_2, m_2)) \rightarrow True/False$. The algorithm takes two sets composed of signatures, rings, and messages as input. If the two signatures are linked, it outputs True, otherwise False.

3. The Improved bLSAG Scheme and Voting Weight Function

3.1. The Improved bLSAG Scheme

In order to hide the identity of voters participating in the redacting consensus without the intervention of a third party, we consider using the linkable ring signature technology used in Monero [16], namely the Back's Linkable Spontaneous Anonymous Group (bLSAG) signatures scheme. In theory, it has strong privacy protection characteristics and can provide a secure basis for the blockchain. However, in our redactable blockchain scheme, each voter is allowed to vote for multiple objects in a round, if the original bLSAG scheme is used directly, the voter will be linked, so we add an information identification *id*, and embed it in the linkable tag and signature to allow voters to vote for different objects in the same round. Next, we will give the specific construction after improvement.

- Setup. Let *E* represent an elliptic curve defined on a finite field $\mathbb{F}_p$, *G* be a generator on *E*, and let *d* represent the order of *G*, where *p* and *d* are sufficiently large prime numbers. Further, we need to define the following two hash functions:

  $$\mathcal{H}_1 : E(\mathbb{F}_p) \rightarrow E(\mathbb{F}_p). \quad \mathcal{H}_2 : \{0,1\}^* \rightarrow \{1, 2, \cdots, d-1\}.$$

  The public parameter is $pp = (\mathbb{F}_p, G, E, d, p, \mathcal{H}_1, \mathcal{H}_2)$, and it is used as an implicit input to other algorithms.
- KeyGen. Each user *u* randomly selects a number k_u, satisfying $0 < k_u < d$ as his secret key, and the corresponding public key $K_u = k_u G$.
- LRSign. Let $m \in \{0,1\}^*$ be the message to sign, and *id* stand for some information identification. Then, the signer chooses some distinct public keys as the ring $R = \{K_1, K_2, \cdots, K_n\}$. Let $1 \leq i \leq n$ represent the signer's secret index in *R*, and the signer's public key is K_i. Signer generates a linkable signature using the following steps:
  1. Compute $L = k_i \mathcal{H}_2(id)\mathcal{H}_1(K_i)$ as a linkable tag.
  2. Select a random number $\gamma \in \{1, 2, \cdots, d-1\}$, and compute

     $$c_{i+1} = \mathcal{H}_2(m, \gamma G, \gamma \mathcal{H}_2(id)\mathcal{H}_1(K_i)).$$

  3. For $j = 1, 2, \cdots, i-1, i+1, \cdots, n$, pick a random number $r_j \in \{1, 2, \cdots, d-1\}$ and compute $c_{j+1} = \mathcal{H}_2(m, r_j G + c_j K_j, r_j \mathcal{H}_2(id)\mathcal{H}_1(K_j) + c_j L)$; where the subscript exceeds *n*, the module *n* is required.
  4. Calculate $r_i = \gamma - c_i k_i (\text{mod } d)$.
  5. The signature is $\sigma(m) = (R, L, c_1, r_1, \cdots, r_n)$ with information identification *id*.
- LRVer. The verifier verifies whether the signature $\sigma(m)$ is a valid signature created by the signer who holds a private key corresponding to a public key in *R* as follows:
  1. Check whether *dL* is equal to 0, if so, then reject.
  2. For $j = 1, 2, \cdots, n$, iteratively calculate

     $$c'_{j+1} = \mathcal{H}_2(m, r_j G + c_j K_j, r_j \mathcal{H}_2(id)\mathcal{H}_1(K_j) + c_j L),$$

     where the subscript exceeds *n*, the module *n* is required.

3. Check whether $c_1' = c_1$, if so, accept; otherwise, reject.

- Link. Given signatures $\sigma' = (R', L', c_1', r_1', \cdots, r_n')$ and $\sigma'' = (R'', L'', c_1'', r_1'', \cdots, r_n'')$, two messages m', m'' and two information identifiers id', id''. The verifier first uses the *LRVer* algorithm to check whether the two signatures are valid. If so, check whether $id' = id'' \wedge L' = L''$. If it holds, it means that the two signatures are from the same signer for the same information identification, and they will be linked, otherwise the two signatures will not be linked.

3.2. Security Analysis

In this section, we will analyze the security of the improved bLSAG scheme. The Elliptic Curve Discrete Logarithm Problem (ECDLP) is a hardness assumption crucial to the security of the scheme, which means that given an elliptic curve E defined on a finite field $\mathbb{F}_p$, a point G of order d on E, and a point P that is a multiple of G, it is difficult to find an integer $a \in [0, d)$ such that $P = aG$. Under the hardness assumption of ECDLP, the improved bLSAG scheme is proved to be correct, existentially unforgeable against the adaptive chosen message attack, signer-ambiguous, and linkable through the following theorems.

Theorem 1 (Correctness)**.** *The improved bLSAG scheme is proved to be correct.*

Theorem 2 (Existential Unforgeability Against Adaptive Chosen Message Attack)**.** *Under the random oracle model and the ECDLP assumption, the improved bLSAG scheme is existentially unforgeable against adaptive chosen message adversaries.*

Corollary 1. *Let $\mathcal{O}_1(R, M)$ be a signature oracle that takes the ring set* R*, composed of* n *public keys and the message* M *as input, and generates a signature σ such that $LRVer(R, M, \sigma) = 1$. An improved bLSAG signature scheme is existentially unforgeable against adaptive chosen plaintext attack if, for any PPT algorithm $\mathcal{A}_1$ with $\mathcal{O}_1$ such that $\mathcal{A}_1^{\mathcal{O}_1}(R) \to (R, M, \sigma)$, for a set* R *of public keys selected by $\mathcal{A}_1$, and (R, M, σ) is not included in the query-response pair to $\mathcal{O}_1$, then it satisfies $LRVer(R, M, \sigma) = 1$ only with negligible probability.*

Theorem 3 (Signer Ambiguity)**.** *The improved bLSAG scheme is signer ambiguous under the random oracle model and Decisional Diffie–Hellman Problem (DDHP).*

Corollary 2. *A improved bLSAG signature scheme is signer ambiguous if, for any PPT algorithm $\mathcal{A}_2$, inputs of any message* M*, set* R *containing* n *public keys, set of private keys $\mathcal{K} = \{sk_1, sk_2, \cdots, sk_t\}$, where the public keys corresponding to these private keys belong to* R*, a valid signature σ on (R, M) is generated by user* i *and for any polynomial* P(k)*, where* k *is the security parameter, if $sk_i \notin \mathcal{K} \wedge 0 \leqslant t < n-1$, we have $Pr[\mathcal{A}_2(M, R, \mathcal{K}, \sigma) \to i] \in (1/(n-t) - 1/P(k), 1/(n-t) + 1/P(k))$, otherwise, we have $Pr[\mathcal{A}_2(M, R, \mathcal{K}, \sigma) \to i] > 1 - 1/P(k)$.*

Theorem 4 (Linkability)**.** *The improved bLSAG scheme is linkable.*

Corollary 3. *Let R_1 and R_2 be two set of* n *public keys. An improved bLSAG signature scheme is linkable if, for all sufficiently large* k*, any $i_1, i_2 \in \{1, \cdots, n\}$, messages M_1, M_2, information identification id_1, id_2, and $\sigma_1 \leftarrow LRSign(sk_{i_1}, id_1, R_1, M_1)$, $\sigma_2 \leftarrow LRSign(sk_{i_2}, id_2, R_2, M_2)$, there exists a PPT algorithm $\mathcal{A}_3$ which outputs 1 or 0 with $Pr[\mathcal{A}_3(R_1, R_2, id_1, id_2, M_1, M_2, \sigma_1, \sigma_2) = 1 \wedge (i_1 \neq i_2 \vee id_1 \neq id_2)] \leqslant \xi(k)$ and $Pr[\mathcal{A}_3(R_1, R_2, id_1, id_2, M_1, M_2, \sigma_1, \sigma_2) = 0 \wedge i_1 = i_2 \wedge id_1 = id_2] \leqslant \xi(k)$, where ξ is a negligible function.*

The complete proofs of the above four theorems are in the Appendix A, where the proof of Theorem 1 is in Appendix A.1, the proof of Theorem 2 is in Appendix A.2, the proof of Theorem 3 is in Appendix A.3, and the proof of Theorem 4 is in Appendix A.4.

3.3. Voting Weight Function

In order to achieve flexible and efficient redacting consensus, we introduce a voting weight function. Each user who wants to participate in the redacting consensus obtains different voting weights by solving puzzles with different target values. The smaller the target value, the greater the difficulty and the greater the weight, which will enable the whole network to reach redacting consensus faster.

Definition 1 (Voting Weight Function $\mathcal{F}$)**.** *A voting weight function $\mathcal{F}$ is a mapping from target value to weight, and its form is as follows, where* v *is the maximum voting weight determined by the system and $i \in \mathbb{N}$.*

$$\mathcal{F}(T_i) = k_i.\ T_i \in \{1, 2, 2^2, \cdots, 2^{256}\}, k_i \in \{1, 2, \cdots, v\}$$

The setting of the voting weight function needs to be combined with the system parameters and constraints, and both it and the target value of the moderate puzzle to select voters are mainly related to the target value T' of the underlying consensus mechanism of the blockchain system. Generally speaking, the target value of the voting weight function is a multiple of T', and the corresponding weight is some small values. For example, the system can set $\mathcal{F}(T_1) = 1, \mathcal{F}(T_2) = 3, T_1 = 116T', T_2 = 78T'$, refer to the parameter settings in Section 5.

4. The Proposed Anonymous and Efficient Redactable Blockchain Scheme

4.1. System Model

The system diagram of our AeRChain scheme is shown in Figure 1. The roles of the AeRChain scheme mainly include users, block producers, and voters. Note that the division of each role is not independent, for example, a user may be a block producer and a voter at the same time.

1. Users. Users are also called ordinary users in the blockchain system and are the main members of the blockchain system. Users can be the sender or receiver of the transaction, they can also become block producers through the underlying consensus mechanism, voters participating in the redacting consensus, or other users unrelated to the transaction.
2. Block Producers. Block producers, also called miners, are the main force for maintaining the blockchain system. They are responsible for collecting transactions and packaging them into new blocks to expand the blockchain. In our AeRChain scheme, block producers are also responsible for collecting and aggregating votes, updating the status value of redacted blocks, and replacing old blocks with new redacted blocks.
3. Voters. Voters are selected from users through a moderate puzzle called custom Proof-of-Work (cPoW), and obtain corresponding voting weights according to the voting weight function. Voters are mainly responsible for anonymous voting on redacting requests sent by users, and are core members of the redacting consensus process.

The main idea of the scheme is that if a user j wants to redact the content of a block $\mathfrak{B}_i$, the user can make a redacting request, including the index i and the redacted content $\mathfrak{B}_i^*$. Other users can vote for the redacting request by solving moderate puzzles cPoW and obtaining voting weights w according to the voting weight function. After that, the voters anonymously vote on the redacting requests within a specified time. When users receive a vote, they verify whether the voting information is correct, and whether the voters are legal. If the verification is passed, the weight value contained in these voting data will be accumulated. When the value reaches a certain threshold ts, it indicates that the whole network has reached a consensus on the redacting request. Then, the block producer aggregates the corresponding proofs, packs them together with ordinary transactions into a new block, and adds the block to the blockchain. At the same time, the block producer is also responsible for updating the content of redacted blocks. Finally, he broadcasts the new chain $\mathfrak{C}'$, then other users update the local chain after receiving and verifying $\mathfrak{C}'$.

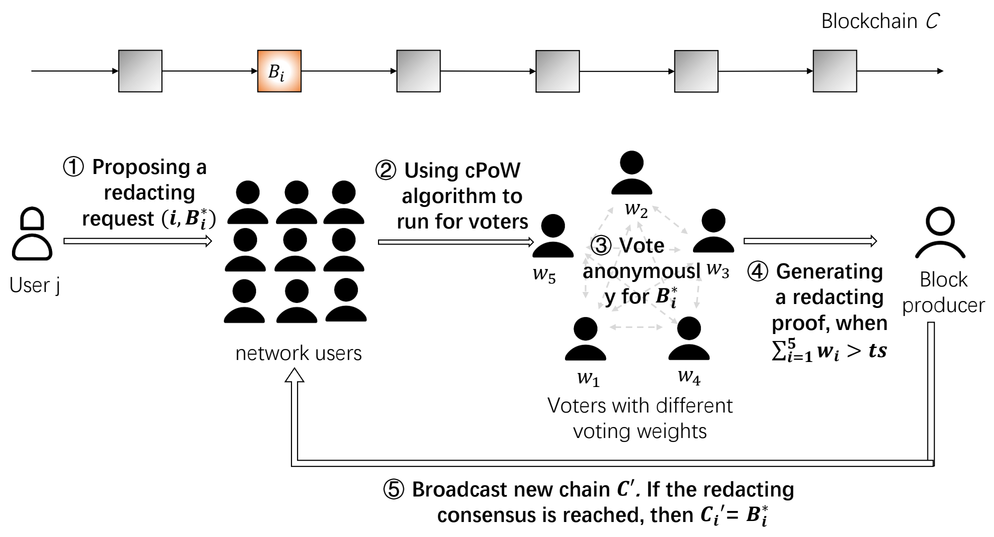
Blockchain C
B_i
① Proposing a redacting request (i, B_i^*)
User j
network users
② Using cPoW algorithm to run for voters
w_1
w_2
w_3
w_4
w_5
③ Vote anonymously for B_i^*
Voters with different voting weights
④ Generating a redacting proof, when $\sum_{i=1}^{5} w_i > ts$
Block producer
⑤ Broadcast new chain C'. If the redacting consensus is reached, then $C_i' = B_i^*$

Figure 1. AeRChain system diagram.

4.2. Threat Model

In our AeRChain scheme, the random algorithm cPoW is used to select voters. Its principle is similar to the underlying PoW consensus, which can ensure that the selected users are the honest majority, but there may still be malicious users among them, so the voters are semi-trusted. Similarly, the miners elected through the underlying PoW consensus are also the honest majority, so they are also semi-trusted. Ordinary users are untrustworthy, and they may jointly launch collusion attacks with malicious voters or malicious miners. In summary, there may be the following threats in the AeRChain scheme: (1) Ordinary users vote for redacting requests and try to pass the voting verification. (2) Malicious voters vote repeatedly or abuse voting rights in an anonymous environment, that is, they vote for the same object multiple times in the same round, or vote in the same round with more weight than they themselves have, and attempt to escape the linkability of anonymous signatures. (3) Malicious voters vote for maliciously redacted transactions and try to get the voting weight to reach the threshold. (4) Other users try to generate anonymous signatures of legal voters, and link pseudo-anonymous signatures with real voter signatures. (5) In the case of a voter performing a legitimate vote, other users attempt to obtain the voter's identity information in order to bribe the voter.

4.3. Detailed Description of the AeRChain Scheme

In order to add a redactable function to the blockchain, we have slightly expanded the block header of the traditional blockchain. Specifically, we add a list variable *oh* in the redactable block to store the historical information of the Merkle Tree root of the block. Because the root value $\mathcal{G}(tx_i)$ of the Merkle tree in the traditional block is implicit in *oh*, we remove it so that the block is now in the form of $\mathfrak{B}_i := (sl_i, ne_i, ph_i, oh_i, tx_i)$. If the block has not been redacted, then oh_i is equal to $\mathcal{G}(tx_i)$. Otherwise, oh_i is a list that stores all the historical Merkle tree root values of the redacted block, i.e., $oh_i = (oh_i^0, oh_i^1, \cdots, oh_i^c)$, and c is the number of times the block has been redacted, oh_i^0 is the initial Merkle tree root of the block, and oh_i^c is the current Merkle tree root value of the block. For convenience of explanation, we assume a block can be redacted at most once, but it is easy to expand to multiple times by changing *oh* to a list.

In addition, the AeRChain scheme uses rounds to promote the execution process of the blockchain. If the voting period is q slots and the network delay is Δ slots, then one round is $q + \Delta$ slots. The specific duration of each round is independent of the block generation time and is determined by the network environment and system parameters.

Initialisation:

When the blockchain system is initialized, the system generates the public parameter *pp* through the *Setup* algorithm and broadcasts it to all users. Each user obtains their public key pk_i and private key sk_i through the *KeyGen* algorithm, and then makes pk_i public, so that any user knows the public keys of other users in the system. When the blockchain is initialized, blockchain $\mathfrak{C}$ has only one genesis block, which contains some necessary information related to the system, including public parameters, system parameters, and user public keys. All users maintain a local blockchain $\mathfrak{C}$ and a list *EditVote* that collects voting information, which is updated regularly.

Generating redacting requests:

When a user wants to redact block $\mathfrak{B}_i$, the user can generate and broadcast the corresponding redacting request, which includs the index *i* of the block to be redacted and the corresponding redacted content $\mathfrak{B}_i^*$ (i.e., candidate block). In addition, each user who requests redacting needs to pay a redacting fee, which is determined by different systems, such as according to the scope and quantity of redacting content. When the whole network reaches a consensus on a redacting request, the redacting fee will be distributed according to the voting weight of voters. Since voters are anonymous, voters can submit relevant proof information of voting and a one-time address for allocating awards after reaching a consensus.

Redacting consensus:

Redacting consensus is the core of all redacting steps, including generating voters, voting for candidate blocks, and updating the blockchain. To prevent adversaries from voting multiple times for the same redacting request in the same voting period, we added a key binding identification, so that if an adversary uses multiple pairs of keys to do this, these votes will not pass the verification. If user *i* wants to participate in an anonymous redacting consensus, *i* first selects a ring member set *R* composed of users' public keys, and then uses the appropriate target value *tv* set by the system to run for voters by solving moderate puzzles we call "custom Proof-of-Work" (cPoW), and vote on the redacting request during the current voting period *q*.

Suppose the current blockchain is $\mathfrak{C}$, the blockchain head is $\mathfrak{B}_l := (sl_l, ph_l, oh_l, ne_l, tx_l)$, and the new block is $B = (sl, ph, mt, tx)$, where *mt* is the Merkle tree root $\mathcal{G}(tx)$ of the new block. The *cPoW* Algorithm 1 takes the new block header $bh := (ph, mt, sl)$, the target value *tv* of the *cPoW*, the voting weight function $\mathcal{F}$, the key binding identification $lid := k_i\mathcal{H}_1(K_i)$, and the ring member set *R* as inputs, where *ph* is the hash value of $\mathfrak{B}_l$, *sl* is the start timestamp of the current round, and K_i and k_i are the public and secret keys of the user calling the algorithm, respectively. Then, the user looks for some random value *ne*, so that the hash value of these data is smaller than *tv*. After successfully finding *ne*, the voter will obtain the voting weight *w* and the corresponding proof *info*, so in this round, he will become a voter and can vote anonymously on the redacting request within the current voting period *q* slots.

Algorithm 1 Custom Proof-of-Work Algorithm *cPoW*

```
Input: the new block header $bh := (ph, mt, sl)$, the target value $tv$, the voting weight function $\mathcal{F}$, the key binding
    identification $lid$, and the ring member set $R$
Output: weight $w$ and proof $info$
1: parse $w := 0$, $info := \varnothing$, the current timestamp is $tm$;
2: while $tm < sl + q + \Delta$ do
3:     if find $ne$ to make $\mathcal{H}(tm, ph, ne, mt, \mathcal{H}(lid||R)) < tv$ then
4:         $w = w + \mathcal{F}(tv)$, $info = info \cup (ph, tm, ne, mt, tv, lid, R)$;
5: return $(w, info)$;
```

The voter generates a voting message *m*, which is composed of information identification *id*, ring member set *R*, voting weight *w*, and proof information *info*, where *id* includes the hash value of a candidate block and the serial number r of the current round. Then, the voter uses the algorithm *LRSign* to sign m and broadcasts the voting information $vm(m,\sigma)$. When other users receive votes from the network, they verify them using the *VerifycPoW*, *LRVer*, and *Link* algorithms, and add the verified vm to *EditVote*. The *VerifycPoW* Algorithm 2 takes the proof information *(w,info)*, the voting weight function $\mathcal{F}$, linkable tag L used in improved bLSAG, and information identification id as the input. The verification process is similar to *cPoW* algorithm, except that the judgment on the correctness of the voter's identity information is added.

Algorithm 2 Verifying cPoW Algorithm *VerifycPoW*

```
Input: weight $w$, proof $info$, the voting weight function $\mathcal{F}$, linkable tag $L$, and information identification $id$
Output: 0 or 1
1: parse $info := (ph, tm, ne, mt, tv, lid, R)$, $c := 0$, the hash value of the chain head at $tm - 1$ slot is $ph'$, the start
   timestamp of round $r$ is $sl$;
2: for each $info$ do
3:     if $\mathcal{H}(tm, ph, ne, mt, \mathcal{H}(lid||R)) < tv \wedge ph = ph' \wedge L = \mathcal{H}_2(id)lid \wedge tm \in [sl, sl + q + \Delta)$ then
4:         $c = c + \mathcal{F}(tv)$;
5: if $c = w$ then
6:     return 1;
7: return 0;
```

When the cumulative weight of votes for a candidate block reaches a certain threshold ts, the block producer aggregates all the proof information contained in the votes into a redacting proof, packs it together with ordinary transactions into a new block $\mathfrak{B}'$, then links it to the blockchain head. At the same time, the block producer replaces the original block $\mathfrak{B}_i$ with the redacted block $\mathfrak{B}_i^*$, and attaches the Merkle tree root of the old block $\mathfrak{B}_i$ to the state variable oh in the block header of $\mathfrak{B}_i^*$. Finally, the new chain $\mathfrak{C}'$ is broadcast. After other users receive it, the algorithm *LRVer*, *VerifycPoW*, and *Link* are used to verify the legitimacy of $\mathfrak{C}'$ and update the local blockchain.

Different systems have different thresholds, which need to exceed the maximum number of votes that malicious users can generate. Because we mainly focus on the process of redacting consensus, the steps for users to verify the validity of candidate blocks and the whole chain after reaching a redacting consensus can be referred to [8]. Note that redacting consensus and ordinary block consensus are parallel, that is to say, users may also become voters in the process of running for block producers, but it depends on whether users want to participate in redacting consensus.

5. Security Analysis and Experimental Evaluation

5.1. Security Analysis

In this section, we analyze the security of the proposed AeRChain scheme, which is mainly based on the hardness of ECDLP and the collision-resistant property of the hash function.

Firstly, the AeRChain scheme with anonymity function satisfies the following security properties:

1. Signer anonymity. Voters use the improved bLSAG scheme to sign their votes, because everyone in the ring has an equal position and can generate a signature; unless the real signer exposes himself, the verifier cannot identify who is the signer. Therefore, this property ensures the identity privacy of voters.
2. Linkability. In our redactable blockchain scheme, the linkable tag is composed of the signer's secret key, the public key, and the voting information identification. If each user has a unique key pair, because the serial number of the voting round is also unique, the linkable tag must be unique. If the user holds multiple key pairs, the key binding identification associated with one of the key pairs needs to be input before calculating the *cPoW* solution, so the key information cannot be changed after obtaining the solution. In other words, each pair of keys corresponds to a puzzle. Only when the puzzle is successfully solved can the corresponding voting weight be

obtained. If the adversary wants to increase the voting weight, a certain number of puzzles must be solved. Therefore, when a voter votes for the same candidate block in the same round with more than the weight he obtains, these votes will be linked and cannot pass the legitimacy verification, so this property prevents repeated voting in the redacting consensus.

3. Unforgeability. On the one hand, the security of the improved bLSAG signature scheme is based on the hardness of ECDLP, which means that for any PPT algorithm $\mathcal{A}$, the probability that $Pr[\mathcal{A}(G, aG) = a]$ is negligible, where $G, aG \in E(\mathbb{F}_p)$. If a voter wants to forge the signature, he needs to obtain the signer's secret key by solving the ECDLP. On the other hand, since the solution of each *cPoW* is bound to a key pair, if the voter wants to change the key pair after finding the solution, he needs to find a hash collision that satisfies the solution. However, as we all know, it is difficult to solve the ECDLP and find a hash collision, and no one can forge or replace the signature except with negligible probability.

Secondly, because only users who successfully solve the *cPoW* will get the right to vote, even if they vote anonymously, the solution of the *cPoW* puzzle is bound to the information of the ring set and key binding identification used for the anonymous signature. Therefore, the votes of ordinary users who fail the election will be rejected by verifiers, meaning that the AeRChain scheme can resist threats (1).

Thirdly, since the key binding identification binds each *cPoW* solution to a pair of key pairs, once the malicious voters with a pair of keys repeatedly vote, their votes will not pass the verification of the *Link* algorithm. For malicious voters with multiple pairs of keys, they can only obtain more voting weight by solving multiple cPoWs. Otherwise, once the key is replaced and the vote is repeated, their vote will not pass the *Link* algorithm, *LRVer* algorithm, and *VerifycPoW* algorithm verification. Thus, by embedding the key binding identification and ring member set into the puzzle of selecting voters, no matter how many pairs of keys a voter has, in an anonymous environment the verifier will reject multiple votes by the same voter for the same object in the same round, or reject votes whose weight do not match the voter's vote weight. This means that the AeRChain scheme can defend against threats (2) and (3). In addition, the AeRChain scheme is resistant to threats (4) and (5) due to its signer anonymity, linkability, and unforgeability shown above.

In summary, based on the security analysis of the improved bLSAG scheme (Section 3.2), and through the combined verification of key binding identification, *VerifycPoW*, *Link*, and *LRVer*, the AeRChain scheme can defend against all threats listed in the threat model (Section 4.2).

Finally, the anonymous and efficient redactable blockchain $\mathfrak{C}$ satisfies the characteristics of chain quality and chain growth, such as ordinary immutable blockchain $\mathfrak{C}'$, and follows the definition of [14]. Because $\mathfrak{C}$ is essentially an extension of $\mathfrak{C}'$, it does not remove blocks from the chain. The *cPoW* algorithm of our scheme is parallel to the *PoW* algorithm of traditional blockchain, that is, the solution of *cPoW* may also be obtained during the process of running *PoW*, which will not affect the basic block consensus or slow down the original chain growth. Therefore, if the immutable blockchain $\mathfrak{C}'$ meets the chain growth property, then $\mathfrak{C}$ will also meet the chain growth property.

In addition, due to the collision-resistant property of the hash function, $\mathfrak{C}$ also satisfies the property of chain quality and the extended definition of common prefix given by [9], i.e., the redactable common prefix, which is suitable for blockchains with a redaction function. The redaction operation is introduced in our scheme, and adversaries may modify the honest block $\mathfrak{B}$ to the malicious block $\mathfrak{B}^*$. However, when an adversary makes a redacting request that includes candidate block $\mathfrak{B}^*$, since the adversary only accounts for μ ($\mu < 1/2$) parts of the total computing power, it can be seen from the "no long block withholding" lemma [17] that, as long as the system sets a reasonable voting weight threshold *ts* and a reasonable voting weight function $\mathcal{F}$, the probability of the adversary successfully redacting block $\mathfrak{B}$ is negligible unless honest users vote for $\mathfrak{B}^*$. In addition, if an adversary wants to construct a candidate block of $\mathcal{H}(\mathfrak{B}^*) = \mathcal{H}(\mathfrak{B})$ to cheat honest users to vote for it, then, due to the collision-resistant property of the hash function, the probability that the adversary can succeed is negligible, where $\mathfrak{B}^* \neq \mathfrak{B}$, $\mathfrak{B}^*$ is malicious content, and $\mathfrak{B}$ is normal content. Therefore, if $\mathfrak{C}'$ satisfies the chain quality property, then $\mathfrak{C}$ will satisfy the chain quality property.

The redactable common prefix means that if the chain $\mathfrak{C}_1$ with the length of l_1 and the chain $\mathfrak{C}_2$ with the length of l_2 are owned by two honest users u_1 and u_2 at slot sl_1 and sl_2, respectively, where $sl_1 \leqslant sl_2$. One of the following two conditions must be satisfied: (1) The prefixes of $\mathfrak{C}_1$ and $\mathfrak{C}_2$ consisting of the first $l_1 - k$ records are identical, where k is a parameter of the common prefix. (2) For each $\mathfrak{B}_i^*$ in the prefix of $\mathfrak{C}_2$ consisting of the first $l_1 - k$ records, but not in the prefix of $\mathfrak{C}_1$ consisting of the first $l_1 - k$ records, it must mean that the whole network has reached a redacting consensus on $\mathfrak{B}_i^*$ at the time of sl_i $(sl_1 < sl_i < sl_2)$, and its proof is stored in the first $l_1 - k$ blocks.

5.2. Parameters Settings

The parameter settings in our scheme refer to [9], and the following constraints and relationships need to be satisfied:

We assume that the number of users in the whole network is n, the proportion of adversaries is $a(a < 1/2)$, and the voting period is t slots. The adversary finds the solutions of the *cPoW* puzzle q slots earlier than honest users. Let $q \approx k * 2^l / T'$, where k is the common prefix parameter, l is the output length of the hash function and T' is the target value for the underlying PoW blockchain. Let α_i and β_i represent the number of cPoWs expected to be solved by honest users and adversaries, respectively, in each slot when they select the target value T_i. Accordingly, N_h and N_a represent the maximum number of *cPoWs* solved by honest users and adversaries in t and $t + q$ slots respectively. We take N_a as the voting threshold ts.

For any $\delta > 0$, with negligible probability $P_1 = exp^{-(\delta * min\{\delta,1\} * (\sum_{i=1}^{max} k_i * \beta_i) * (q+t))/3}$, it holds that $N_a > (1+\delta) * (\sum_{i=1}^{max} k_i * \beta_i) * (q + t)$, where k_i is the voting weight corresponding to T_i. For any $\delta \in (0,1)$, with negligible probability $P_2 = exp^{-(\delta^2 * (\sum_{i=1}^{max} k_i * \alpha_i) * t)/2}$, it holds that $N_h \leqslant (1-\delta) * (\sum_{i=1}^{max} k_i * \alpha_i) * t$, because we need legitimate voters who can solve cPoW puzzles to be an honest majority, so we need to guarantee $N_h > N_a$. Then, we assume that $T_i = x_i * T_{max}$, $1 \leqslant i < max$, $k_j = y_j * k_1$, $1 < j \leqslant max$, where x_i and y_j are two intermediate variables, x_i represents the proportion between each target value and the minimum target value, and y_j represents the proportion between each voting weight and the maximum voting weight. The relevant parameters are substituted into $N_h > N_a$ and reduced to the following formula:

$$t > q/(((1-a)(1-\delta)(x_1 h_1 + \sum_{i=2}^{max-1} x_i y_i h_i + y_{max} h_{max}))$$
$$/(a(1+\delta)(x_1 a_1 + \sum_{i=2}^{max-1} x_i y_i a_i + y_{max} a_{max})) - 1).$$

According to the above constraints, and for generality, we finally set these parameters as follows: $a = 0.4$, voting weight function $\mathcal{F}(T_1) = 1, \mathcal{F}(T_2) = 3$, $\delta = 0.1$, and set $h_1 = 0.2$, $h_2 = 0.8$, $a_1 = 0.6, a_2 = 0.4$, then $t > 1.7q$, so we set $q = 10, t = 18$. If $P_1 = exp^{-15}$, $P_2 = exp^{-14}$, then $T_1 = 116T', T_2 = 78T'$, and the voting threshold ts is approximately 4950, this value is the cumulative weight of different votes, so the total number of votes is less than this value, and these votes are cast during the voting period of 18 slots.

5.3. Experimental Evaluation

We used Python language to construct a simplified blockchain system based on PoW consensus, which simulates some basic functions of Bitcoin. Then, in order to realize the redactable function of the blockchain based on voting consensus, we added a list variable *oh* in the block header to store the old Merkle root value of the block. Our experimental environment is an Ubuntu 16.04 (64bits) system, and the relevant configuration is AMD Ryzen 5 3600 6-Core Processor with 3.6 GHz and 8 GB of RAM. Next, we conducted a series of experiments and evaluations based on the constructed blockchain. Because our scheme utilizes the improved bLSAG scheme and cPoW algorithm, the difference in experimental results is mainly related to the number of ring members and the target values.

First, we evaluated the average time required to verify a vote, as shown in Figure 2a. It can be seen from the figure that even if the number of ring members reaches 50, the time for verifying votes is acceptable. For example, a ring with 11 members is currently recommended in Monero. Therefore, the cost of verifying votes is negligible relative to the whole redacting consensus process.

Secondly, we evaluated the average size of a vote, as shown in Figure 2b. It can be seen from the figure that even though there are many ring members, the size of the vote is small relative to the network communication load.

Thirdly, as shown in Figure 2c, we evaluated the time required for our scheme to reach a redacting consensus under the different number of ring members when each block contains a different number of transactions, and compared the results with scheme [9]. In the figure, we use "Li" to represent [9] scheme, "RN" represents the number of ring members and shows the time required to reach a redacting consensus when the number of ring members is two, four, six, and eleven. On the one hand, the time of our scheme is far less than the 513 slots of [8]. On the other hand, in our scheme, under the average case, as long as each vote contains no more than six ring members, the time to reach a redacting consensus is faster than Li et al. When the number of ring members exceeds seven, taking the number of eleven ring members currently recommended by Monero as an example, our scheme is at most two slots slower than Li et al. Therefore, when the number of ring members is reasonable, compared with the existing voting-based efficient redactable permissionless blockchain scheme, our scheme can reach anonymous redacting consensus at a faster speed.

Finally, we evaluated the size of the redactable blockchain in the average case according to the different percentages of the deleted block data in the whole chain. As shown in Figure 2d, we fixed the length of the chain to 1000 blocks, where each block contains 3000 transactions, and set the number of ring members to eleven as required by the current Monero. The size of an ordinary immutable blockchain is about 0.96 GB. After using AeRChain, with the deletion of more and more block data, the size of the redactable blockchain will be much smaller than that of an ordinary blockchain, which can reduce the communication traffic in the network.

As redacting data in the blockchain is an important operation, redacting operations should only be allowed under special circumstances, and data cannot be redacted frequently. Therefore, an efficient redacting scheme should specify the maximum number of times that data can be redacted within a period, and at the same time make the time of reaching a redacting consensus within the user's acceptable range, such as a few hours. Although the improved bLSAG scheme seems to bring some overhead compared with ordinary signatures, compared with the permissioned blockchain, there are usually only a few redacting requirements in the permissionless blockchain. In our scheme, the overhead is mainly determined by the number of ring members. Therefore, as long as the system specifies or recommends a reasonable number of ring members to users according to the actual application scenario, the redacting consensus is efficient and the overhead is acceptable.

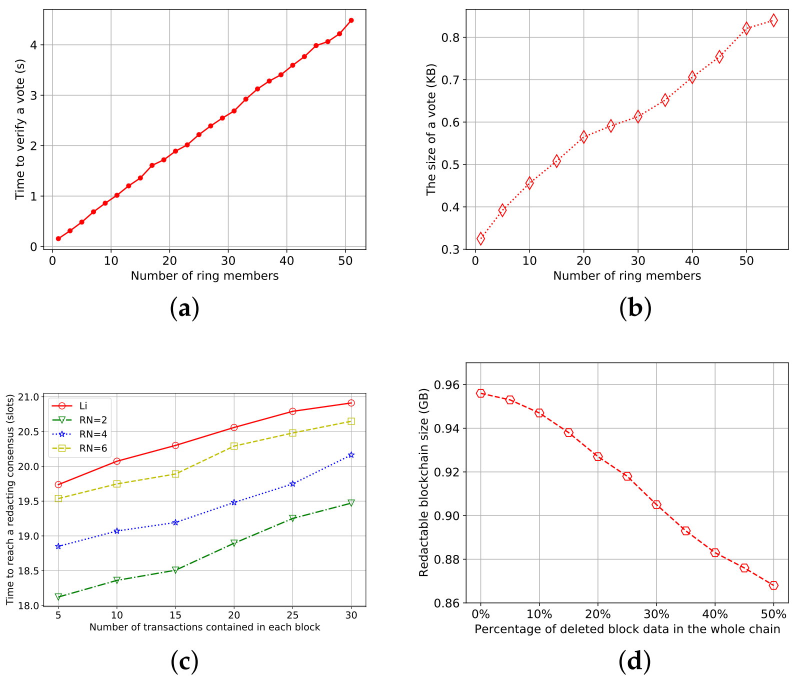

Figure 2. Experiment of AeRChain scheme. (**a**) Time to verify a vote; (**b**) The size of a vote; (**c**) Time to reach a redacting consensus; (**d**) Redactable blockchain size.

6. Conclusions

In this paper, we propose an anonymous and efficient redactable blockchain scheme based on PoW in a permissionless setting, called "AeRChain". Firstly, we improve the bLSAG scheme and use it to hide the identity of voters participating in the redacting consensus. Secondly, we introduce a voting weight function and a moderate puzzle with variable target values to improve the efficiency of redacting consensus. Finally, the experimental results show that, compared with existing schemes, our scheme can achieve efficient and anonymous redacting consensus with a low overhead, and can reduce communication traffic.

Author Contributions: Conceptualization, B.L.; methodology, B.L.; software, B.L.; validation, B.L.; formal analysis, B.L.; investigation, B.L.; resources: B.L.; writing—original draft preparation, B.L.; supervision and instructor: C.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by Natural Science Foundation of Guangdong Province of China (2021A515010857, 2022A1515010029).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: We are very grateful for the hardware facilities provided by the College of Information Science and Technology of Jinan University and Pazhou Laboratory and their full support. In addition, we also thank the reviewers for their precious time.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Appendix A.1. Proof of Theorem 1

Proof. Because for $j = 1, 2, \cdots, n$, c'_{j+1} is computed iteratively, and finally c'_1 is obtained by modulo operation. When $j \neq i$, c'_j is calculated from the previous value c'_{j-1} and $c'_j = c_j$. When $j = i$, since $r_i = \gamma - c_i k_i \pmod{d}$,

$$r_i G + c_i K_i = (\gamma - c_i k_i)G + c_i K_i = \gamma G$$

and

$$\begin{aligned} r_i \mathcal{H}_2(id)\mathcal{H}_1(K_i) + c_i L &= (\gamma - c_i k_i)\mathcal{H}_2(id)\mathcal{H}_1(K_i) + c_i k_i \mathcal{H}_2(id)\mathcal{H}_1(K_i) \\ &= \gamma \mathcal{H}_2(id)\mathcal{H}_1(K_i) \end{aligned}$$

so

$$\begin{aligned} c'_{i+1} &= \mathcal{H}_2(m, r_i G + c_i K_i, r_i \mathcal{H}_2(id)\mathcal{H}_1(K_i) + c_i L) \\ &= \mathcal{H}_2(m, \gamma G, \gamma \mathcal{H}_2(id)\mathcal{H}_1(K_i)) \\ &= c_{i+1} \end{aligned} \tag{A1}$$

Therefore, at the end of the iterative calculation, $c'_1 = c_1$, theorem 1 is proved. □

Appendix A.2. Proof of Theorem 2

Proof. The following proof procedure will incorporate the Rewind-on-Success lemma [18]. Let R be a list of public keys, each of which is generated according to the *KeyGen* algorithm of the improved bLSAG scheme, and *ID* is the information identification set. $\mathcal{H}_1$ and $\mathcal{H}_2$ are independent random oracles that generate random outputs, and if repeated queries are made, $\mathcal{H}_1$ and $\mathcal{H}_2$ return the same result. $\mathcal{O}_s$ is a signature oracle that returns a valid signature based on the input query, $\mathcal{O}_s$ also queries $\mathcal{H}_1$ and $\mathcal{H}_2$, and its query is consistent with the query to $\mathcal{O}_s$. The signature output by $\mathcal{O}_s$ looks exactly like the signature that the real signer actually signed. For some independent random oracles $\mathcal{H}_1$ and $\mathcal{H}_2$, and for the security parameter k, P_1 is some polynomials that are not greater than the polynomial growth.

Assuming a Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$ conducts at most $Q_{\mathcal{H}}$ queries on the combination of $\mathcal{H}_1$ and $\mathcal{H}_2$, and at most $Q_{\mathcal{O}_s}$ queries on $\mathcal{O}_s$, and can forge improved (1,n)-bLSAG signatures with a non-negligible probability, that is,

$$Pr[\mathcal{A}(R) \to (m, \sigma) : \mathcal{V}er(R, m, \sigma) = 1] > 1/P_1(k)$$

First, construct a PPT simulator $\mathcal{S}$ that invokes $\mathcal{A}$ and solves the ECDLP of at least one public key belonging to R with non-negligible probability. Let $U = \{K_1, K_2, \cdots, K_n\}$ denote a subset of R, id represents an information identification in *ID*, and denote by $\sigma = (U, L, c_1, r_1, \cdots, r_n)$ that the forged signature on U and id satisfies the *LRVer* involving the following equations.

$$c_1 = \mathcal{H}_2(m, r_n G + c_n K_n, r_n \mathcal{H}_2(id)\mathcal{H}_1(K_n) + c_n L),$$
$$c_{i+1} = \mathcal{H}_2(m, r_i G + c_i K_i, r_i \mathcal{H}_2(id)\mathcal{H}_1(K_i) + c_i L),$$

where $L = k_0 \mathcal{H}_2(id)\mathcal{H}_1(K_0)$, $1 \leqslant i \leqslant n-1$, and G is consistent with the parameter in the *Setup* algorithm. $\mathcal{S}$ will call $\mathcal{A}$ with constructed inputs, receive and process $\mathcal{A}$'s outputs, and may call $\mathcal{A}$ multiple times based on the outputs of previous calls to $\mathcal{A}$. Additionally, $\mathcal{S}$ flips coins for $\mathcal{H}_1$ and $\mathcal{H}_2$ while recording queries to these oracles. Every invocation of $\mathcal{A}$ is now recorded on a simulation transcript tape, some of which transcripts generate successful forged signatures and some do not. Given any message m, any set of public keys $U = \{K_1, K_2, \cdots, K_n\}$, any information identification id, $\mathcal{O}_s$ will generate a signature. Without knowing any secret key, $\mathcal{S}$ generates the signature by simulating $\mathcal{O}_s$, but back patching $\mathcal{H}$ in the following way. Assuming that for $1 \leqslant i \leqslant n$, $\mathcal{H}_1(K_i)$ has been queried before, it is simulated by $\mathcal{S}$ by randomly choosing $r^1, r^2, \cdots, r^n$ and outputting $\mathcal{H}_1(K_i) = r^i G$. To simulate $\mathcal{O}_s$, $\mathcal{S}$ randomly selects $x \in \{1, 2, \cdots, n\}$, $c_1, c_2, \cdots, c_n$, $r_1, r_2, \cdots, r_n$, and compute $L = r^x \mathcal{H}_2(id)\mathcal{H}_1(K_x)$. Back patch to

$$\begin{aligned} \mathcal{H}_2(m, r_i G + c_i K_i, r_i \mathcal{H}_2(id) r^i G + c_i L) &= \mathcal{H}_2(m, r_i G + c_i K_i, r_i \mathcal{H}_2(id) r^i G + c_i r^x \mathcal{H}_2(id)\mathcal{H}_1(K_x)) \\ &= c_{i+1} \end{aligned}$$

where $1 \leqslant i \leqslant n$ and $i = 1$ after modular operation when $i = n + 1$. Let $\mathcal{E}$ be an event corresponding to each of n *LRVer* queries being included in $\mathcal{A}$'s $Q_{\mathcal{H}}$ queries to a random oracle, or in $Q_{\mathcal{S}}$ signing queries on behalf of $\mathcal{A}$ by the signing oracle. In event $\mathcal{E}'$, in order to verify $\mathcal{A}$'s forged signature, $\mathcal{S}$ needs to flip additional coins, and the probability that c_1 satisfies the *LRVer* is at most $1/(d - Q_{\mathcal{H}} - nQ_{\mathcal{S}})$, where d is a parameter in *Setup*. So we have

$$\begin{aligned} 1/P_1(k) &< Pr[\mathcal{E}]Pr[\mathcal{A}\, forges|\mathcal{E}] + Pr[\mathcal{E}']Pr[\mathcal{A}\, forges|\mathcal{E}'] \\ &\leqslant Pr[\mathcal{E}]Pr[\mathcal{A}\, forges|\mathcal{E}] + 1/(d - Q_{\mathcal{H}} - nQ_{\mathcal{S}}), \end{aligned}$$

and

$$Pr[\mathcal{E}\, and\, \mathcal{A}\, forges] > 1/P_1(k) - 1/(d - Q_{\mathcal{H}} - nQ_{\mathcal{S}}).$$

Since $1/(d - Q_{\mathcal{H}} - nQ_{\mathcal{S}})$ is so small as to be negligible, the probability that $\mathcal{A}$ outputs a forged signature that has already queried the random oracle for the n queries used in *LRVer* is greater than $1/P_1(k)$. Therefore, in each $\mathcal{A}$ transcript that successfully generates a valid signature, there are n queries to $\mathcal{H}_2$, denoted as $Q_{x1}, Q_{x2}, \cdots, Q_{xn}, 1 \leqslant x1 < \cdots < xn$, so that the n queries in *LRVer* match these n queries. $\mathcal{O}_s$ has negligible impact on the queries of the random oracles $\mathcal{H}_1$ and $\mathcal{H}_2$ in response to $\mathcal{A}$'s signature request.

In a signature σ successfully forged by $\mathcal{A}$, let $Q_{x1}, Q_{x2}, \cdots, Q_{xn}, 1 \leqslant x1 < \cdots < xn$ denote the first occurrence of each query in *LRVer* such that the gap x of σ satisfies the following formula.

$$Q_{xn} = \mathcal{H}_2(m, r_{x-1}G + c_{x-1}K_{x-1}, r_{x-1}\mathcal{H}_2(id)\mathcal{H}_1(K_{x-1}) + c_{x-1}L)$$

If $x1 = f$, the signature σ successfully forged by $\mathcal{A}$ is *(f,x)*-forgery. That is to say, all the queries related to *LRVer* appear for the first time at the f-th query, including the query $\mathcal{K}$ on behalf of $\mathcal{A}$ to the random oracle. Then there are f and x, $1 \leqslant f \leqslant Q_{\mathcal{H}}, 1 \leqslant x \leqslant n$, so that $Pr[\mathcal{A}\, produces\, (f,x) - forgery] \geqslant 1/(nQ_{\mathcal{H}}P_1(k) + n^2Q_{\mathcal{K}}P_1(k))$.

Next, for each value of f and x, $\mathcal{S}$ will do a rewind-simulation. First $\mathcal{S}$ calls $\mathcal{A}$ to obtain its Turing transcript T and output. In polynomial time, $\mathcal{S}$ uses both to determine whether it is a successful *(f,x)*-forgery. If yes, continue execution, otherwise abort. T is rewound to the f-th query, and $\mathcal{A}$ performs a rewind-simulation to generate transcript T', where the codes of T and T' in $\mathcal{A}$ are the same, and the common f-th query of T and T' is denoted as $\mathcal{H}_2(m, uG, v\mathcal{H}_2(id)\mathcal{H}_1(K_0))$. Let $\mathcal{C}$ denote the common prefix of T and T'. For queries after f, the flips of new coins will be independent of the flips of coins in T and maintain the consistency of previous queries. $\mathcal{S}$ only knows uG and $v\mathcal{H}_2(id)\mathcal{H}_1(K_0)$ when rewinding, but not u and v. After $\mathcal{A}$ returns the result of the rewind simulation, $\mathcal{S}$ computes the ECDLP for K_x.

Let $\varepsilon_{f,x}(\mathcal{C}) = Pr[T(f,x)|\mathcal{C}] = Pr[T'(f,x)|\mathcal{C}]$, Because $Pr[\mathcal{C}] = \sum_{T\, prefixes} Pr[T]$, $\sum_{f,x}\sum_{C}\varepsilon_{f,x}(C) \geqslant 1/P_1(k)$ and $Pr[T'\, prefixes] = (\varepsilon_{f,x}(\mathcal{C})Pr[\mathcal{C}])/(\sum_{C}\varepsilon_{f,x}(C)Pr[\mathcal{C}])$. Then

$$\begin{aligned} Pr[T'(f,x)] &= \sum_{C} Pr[T'\, prefixes]Pr[T'(f,x)|T'\, prefixes] \\ &= (\varepsilon_{f,x}(\mathcal{C})Pr[\mathcal{C}])/(\sum_{C}\varepsilon_{f,x}(C)Pr[\mathcal{C}]) \cdot \varepsilon_{f,x}(\mathcal{C}) \\ &= \langle \varepsilon^2_{f,x}(\mathcal{C})\rangle / \langle \varepsilon_{f,x}(\mathcal{C})\rangle \\ &\geqslant \langle \varepsilon_{f,x}(\mathcal{C})\rangle \end{aligned}$$

Two *(f,x)* -forgery signatures are generated by rewind simulation tape T' and tape T, where

$$\begin{aligned} &T: \ uG = (r_xG + c_xK_x)\, mod\, d_x = (r_x + c_xk_x)G\, mod\, d_x \\ &T': \ uG = (r'_xG + c'_xK_x)\, mod\, d_x = (r'_x + c'_xk_x)G\, mod\, d_x \\ &T: \ v\mathcal{H}_2(id)\mathcal{H}_1(K_0) = r_x\mathcal{H}_2(id)\mathcal{H}_1(K_x) + c_xr^x\mathcal{H}_2(id)\mathcal{H}_1(K_x)\, mod\, d_0 \\ &T': \ v\mathcal{H}_2(id)\mathcal{H}_1(K_0) = r'_x\mathcal{H}_2(id)\mathcal{H}_1(K_x) + c'_xr^x\mathcal{H}_2(id)\mathcal{H}_1(K_x)\, mod\, d_0, \end{aligned}$$

then the following formula can be obtained:

$$\begin{aligned} k_x &= (r_x - r'_x)/(c'_x - c_x)\, mod\, d_x \\ r^x &= (r_x - r'_x)/(c'_x - c_x)\, mod\, d_0. \end{aligned}$$

$\mathcal{S}$ solves for k_x according to the above formula, and there exists *(f,x)* such that $\langle \varepsilon_{f,x}(\mathcal{C}) \rangle \geqslant 1/(n(Q_{\mathcal{H}} + nQ_{\mathcal{S}})P_1(k))$. Thus, denote $\mathcal{Q}_n = Q_{\mathcal{H}} + nQ_{\mathcal{S}}$, for all possible *(f,x)*, where $1 \leqslant f \leqslant \mathcal{Q}_n, 1 \leqslant x \leqslant n$, $\mathcal{S}$ obtains a solution with the above probability at least once during the run. The complexity of $\mathcal{A}$ is not less than $1/(n\mathcal{Q}_n)$ times that of $\mathcal{S}$, and the probability of success of $\mathcal{S}$ is not negligible, exceeding $1/(n\mathcal{Q}_nP_1(k))^2$. Therefore, the improved bLSAG scheme is existentially unforgeable against adaptive chosen message attacks, and the proof of Theorem 2 is completed. □

Appendix A.3. Proof of Theorem 3

Proof. DDHP means that given four elements $G, aG, bG, cG \in E(\mathbb{F}_p)$, it determines whether $c = ab$. That is, it is impossible for adversaries to distinguish between (aG, bG, cG) and (aG, bG, abG) in PPT. Formally speaking, suppose there is a PPT adversary $\mathcal{A}$, which takes message m, ring R containing n public keys, secret keys set $\mathcal{K} = \{sk_1, sk_2, \cdots, sk_t\}$, $0 \leqslant t \leqslant n-2$, information identification id and a valid signature $\sigma_i(m, R, id)$ signed by signer i as its input, where the public keys corresponding to those secret keys in $\mathcal{K}$ belong to R. $\mathcal{A}$ outputs an index of the actual signer it thinks is in R. In the random oracle model, $\mathcal{A}$ can query $\mathcal{H}_2$ for a maximum of q_2 times.
In the case of $sk_i \notin \mathcal{K} \wedge (0 \leqslant t \leqslant n-2)$, for some polynomial $P(k)$ about the safety parameter k, with probability

$$1/(n-t) - 1/P(k) \leqslant Pr[\mathcal{A}(m, R, id, \mathcal{K}, \sigma_i(m, R, id)) \rightarrow i] \leqslant 1/(n-t) + 1/P(k)$$

Now construct a PPT $\mathcal{C}$ running the following algorithm, which will solve the DDHP together with $\mathcal{A}$. $\mathcal{C}$ takes the number of ring members n, id, message m, and an instance (α, β, δ) of DDHP as input. Randomly select the index i of a signer. Let the secret key of i be $sk_i = \alpha\beta$, the corresponding public key is $pk_i = sk_iG = \alpha\beta G$, and let $v = \mathcal{H}_2(id)$. For all $j \in \{1, \cdots, i-1, i+1, \cdots, n\}$, select some random numbers $sk_j \in \{1, \cdots, d-1\}$, and let $pk_j = sk_jG$, $R = \{pk_1, \cdots, pk_n\}$. For all $j \in \{1, \cdots, n\}$, select some random numbers $x_j, r_j, c_j \in \{1, \cdots, d-1\}$, and let $\mathcal{H}_1(pk_j) = x_jG$ and $\mathcal{H}_2(m, r_jG + c_jpk_j, r_jv\mathcal{H}_1(pk_j) + c_j\alpha\beta v\mathcal{H}_1(pk_i)) = c_{j+1}$, where $c_{n+1} = c_1$. Randomly select t $k_j, (j \neq i)$ to generate $\mathcal{K}$ and signature $\sigma_i(m, R, id) = (R, \delta\mathcal{H}_1(pk_i), c_1, r_1, \cdots, r_n)$. Finally, $\mathcal{C}$ outputs $(R, m, \mathcal{K}, \sigma_i)$.
After the above algorithm, $\mathcal{C}$ gives the output result to $\mathcal{A}$, and then $\mathcal{A}$ guesses the identity of the signer. Firstly, the event that $h_j = \mathcal{H}_2(m, r_jG + c_jpk_j, r_jv\mathcal{H}_1(pk_j) + c_j\alpha\beta v\mathcal{H}_1(pk_i))$ appears in $\mathcal{C}$ and at least once appears in q_2 queries is denoted as E_1, with probability:

$$Pr[Col] = Pr[\cup_{j\in\{1,\cdots,n\},all\, h_j} E_1] \leqslant n \mid E(\mathbb{F}_p) \mid \times q_2/|E(\mathbb{F}_p)|^2 < nq_2/2^k.$$

Therefore, $Pr[\mathcal{A}(m, R, id, \mathcal{K}, \sigma_i(m, R, id)) \rightarrow i \wedge \overline{Col}] > 1/(n-t) + \varepsilon(k) - nq_2/2^k$, where $\varepsilon(k) - nq_2/2^k$ is not negligible about k.

Secondly, $\mathcal{A}$ returns an integer $j \in \{1, \cdots, n\}$ to $\mathcal{C}$. If $j = i$, $\mathcal{C}$ outputs 1, otherwise 0. Then

$$Pr[\mathcal{C} = b] = 1/2 \times Pr[\mathcal{C} = b|b = 0] + 1/2 \times Pr[\mathcal{C} = b|b = 1].$$

If $b = 1$, then $(G, \alpha G, \beta G, \delta G)$ is a DDHP instance, $\sigma_i(m, R, id)$ is a valid signature, and $\mathcal{A}$ will guess the signer from $n-t$ numbers. Then

$$\begin{aligned} Pr[\mathcal{C} = b|b = 1] &\geqslant Pr[\mathcal{C} = b \wedge \overline{Col}|b = 1] \\ &= Pr[\mathcal{A} = i \wedge \overline{Col}|b = 1] + 1/2 \times Pr[\mathcal{A} \neq i \wedge \overline{Col}|b = 1] \\ &= (1 + \varepsilon(k) - nq_2/2^k)/2 + 1/(2n - 2t) \end{aligned}$$

If $b = 0$, then $(G, \alpha G, \beta G, \delta G)$ is not necessarily a DDHP instance, and whether $\sigma_i(m, R, id)$ is a valid signature cannot be determined. Then

$$\begin{aligned} Pr[\mathcal{C} = b|b = 0] &\geqslant Pr[\mathcal{C} = b \wedge \overline{Col}|b = 0] \\ &= 1/2 \times Pr[\mathcal{A} \neq i \wedge \overline{Col}|b = 0] \\ &= 1/2 - 1/(2n - 2t) \end{aligned}$$

Therefore, $Pr[\mathcal{C} = b] \geqslant (2 + \varepsilon(k) - nq_2/2^k)/4$. Because $\varepsilon(k) - nq_2/2^k$ is not negligible, the above probability is better than random guess, which contradicts with DDHP. So the improved bLSAG scheme is signer ambiguous under the random oracle model and DDHP. □

Appendix A.4. Proof of Theorem 4

Proof. Given two information identities id' and id'', two valid signatures $\sigma' = (R', L', c'_1, r'_1, \cdots, r'_n)$ and $\sigma'' = (R'', L'', c''_1, r''_1, \cdots, r''_n)$, which are signatures of two different messages m' and m'' respectively. Since $L' = k_i \mathcal{H}_2(id')\mathcal{H}_1(K_i)$ and $L'' = k_j \mathcal{H}_2(id'')\mathcal{H}_1(K_j)$, if $L' = L''$, that is,

$$k_i \mathcal{H}_2(id')\mathcal{H}_1(K_i) = k_j \mathcal{H}_2(id'')\mathcal{H}_1(K_j) \tag{A2}$$

Assume that each user of the system has only one pair of keys, because *id* is an information identifier related to a specific scenario, for example, in a scenario where voters can only vote for one candidate at most, *id* can represent the round number when voting. In the scenario where voters are allowed to vote for multiple candidates, the *id* can be composed of the round number and the candidate's serial number when voting. Therefore, it is obvious that the Equation (A2) will only hold if $i = j$ and $id' = id''$, which means that the two signatures come from the same malicious signer. Although the malicious signer forges two valid signatures by choosing two different rings, the verifier can link the two signatures through the linkable tag L and information identification *id*, so the improved bLSAG scheme has linkability. If the malicious signer has multiple pairs of keys, in order to prevent the malicious signer from using multiple pairs of keys to forge valid signatures, and then vote for the same candidate multiple times in the same voting period, we consider adding a key binding identification in the process of electing voters, so as to bind each pair of keys with the proof of voting rights, see Section 4 for details. □

References

1. Tanwar, S.; Parekh, K.; Evans, R. Blockchain-based electronic healthcare record system for healthcare 4.0 applications. *J. Inf. Secur. Appl.* **2020**, *50*, 102407. [CrossRef]
2. Hou, H. The application of blockchain technology in E-government in China. In Proceedings of the ICCCN, Vancouver, BC, Canada, 31 July–3 August 2017; pp. 1–4.
3. Matzutt, R.; Hiller, J.; Henze, M.; Ziegeldorf, J.H.; Müllmann, D.; Hohlfeld, O.; Wehrle, K. A quantitative analysis of the impact of arbitrary blockchain content on bitcoin. In Proceedings of the International Conference on Financial Cryptography and Data Security, Nieuwpoort, Curacao, 26 February–2 March 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 420–438.
4. Tatar, U.; Gokce, Y.; Nussbaum, B. Law versus technology: Blockchain, GDPR, and tough tradeoffs. *Comput. Law Secur. Rev.* **2020**, *38*, 105454. [CrossRef]
5. Ateniese, G.; Magri, B.; Venturi, D.; Andrade, E. Redactable blockchain–or–rewriting history in bitcoin and friends. In Proceedings of the EuroS&P, Paris, France, 26–28 April 2017; pp. 111–126.
6. Derler, D.; Samelin, K.; Slamanig, D.; Striecks, C. Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. 2019. Available online: https://eprint.iacr.org/2019/406 (accessed on 20 November 2020).
7. Xu, S.; Ning, J.; Ma, J.; Huang, X.; Deng, R.H. K-time modifiable and epoch-based redactable blockchain. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 4507–4520. [CrossRef]
8. Deuber, D.; Magri, B.; Thyagarajan, S.A.K. Redactable blockchain in the permissionless setting. In Proceedings of the IEEE S&P, San Francisco, CA, USA, 20–22 May 2019; pp. 124–138.
9. Li, X.; Xu, J.; Yin, L.; Lu, Y.; Tang, Q.; Zhang, Z. Escaping from Consensus: Instantly Redactable Blockchain Protocols in Permissionless Setting. 2021. Available online: https://ia.cr/2021/223 (accessed on 10 December 2021).
10. Thyagarajan, S.A.K.; Bhat, A.; Magri, B.; Tschudi, D.; Kate, A. Reparo: Publicly verifiable layer to repair blockchains. In Proceedings of the FC, Seoul, Republic of Korea, 13–17 July; Springer: Berlin/Heidelberg, Germany, 2021; pp. 37–56.
11. Cai, X.; Ren, Y.; Zhang, X. Privacy-protected deletable blockchain. *IEEE Access* **2019**, *8*, 6060–6070. [CrossRef]
12. Ren, Y.; Cai, X.; Hu, M. Privacy-Preserving Redactable Blockchain for Internet of Things. *Secur. Commun. Netw.* **2021**, *2021*, 4485311. [CrossRef]
13. Panwar, G.; Vishwanathan, R.; Misra, S. ReTRACe: Revocable and Traceable Blockchain Rewrites using Attribute-based Cryptosystems. In Proceedings of the SACMAT, Virtual Event, Spain, 16–18 June 2021; pp. 103–114.
14. Garay, J.; Kiayias, A.; Leonardos, N. The bitcoin backbone protocol: Analysis and applications. In Proceedings of the Eurocrypt, Sofia, Bulgaria, 26–30 April 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 281–310.
15. David, B.; Gaži, P.; Kiayias, A.; Russell, A. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Proceedings of the Eurocrypt, Tel Aviv, Israel, 29 April–3 May 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 66–98.
16. Alonso, K.M. Zero to Monero. 2020. Available online: https://web.getmonero.org/de/library/Zero-to-Monero-1-0-0.pdf (accessed on 5 January 2021).

17. Pass, R.; Seeman, L.; Shelat, A. Analysis of the blockchain protocol in asynchronous networks. In Proceedings of the Eurocrypt, Paris, France, 30 April–4 May 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 643–673.
18. Liu, J.K.; Wei, V.K.; Wong, D.S. Linkable spontaneous anonymous group signature for ad hoc groups. In Proceedings of the ACISP, Sydney, Australia, 13–15 July 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 325–335.