Hindawi
Security and Communication Networks
Volume 2021, Article ID 3680359, 20 pages
https://doi.org/10.1155/2021/3680359

Research Article

Fine-Grained and Controllably Redactable Blockchain with Harmful Data Forced Removal

Huiying Hou,[1] **Shidi Hao**,[1] **Jiaming Yuan**,[2] **Shengmin Xu**,[3] **and Yunlei Zhao**[1]

[1]*College of Computer Science and Technology, Fudan University, Shanghai 200433, China*
[2]*College of Computer and Information Science, University of Oregon, Eugene, OR, USA*
[3]*School of Information Systems, Singapore Management University, Singapore*

Correspondence should be addressed to Yunlei Zhao; ylzhao@fudan.edu.cn

Received 13 April 2021; Revised 26 April 2021; Accepted 11 May 2021; Published 29 May 2021

Academic Editor: Yinghui Zhang

Copyright © 2021 Huiying Hou et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Notoriously, immutability is one of the most striking properties of blockchains. As the data contained in blockchains may be compelled to redact for personal and legal reasons, immutability needs to be skillfully broken. In most existing redactable blockchains, fine-grained redaction and effective deletion of harmful data are mutually exclusive. To close the gap, we propose a fine-grained and controllably redactable blockchain with harmful data forced removal. In the scheme, the originator of the transaction has fine-grained control over who can perform the redaction and which portions of the transaction can be redacted. The redaction transaction is performed after collecting enough votes from miners. All users can provide the index of the block containing the harmful data to receive rewards, which are borne by the malicious user who initially posted the data. Miners can forcibly remove the harmful data based on the index. The malicious user will be blacklisted if the reward is not paid within a period of time, and any transaction about such user will not be performed later. In addition, the scheme supports the redaction of additional data and unexpended transaction output (UTXO) simultaneously. We demonstrate that the scheme is secure and feasible via formal security analysis and proof-of-concept implementation.

1. Introduction

The first application of blockchains is Bitcoin [1, 2], which has revolutionized the financial industry. Ever since, hundreds of such cryptocurrencies rise which do not rely on a central trusted authority. The applications of blockchains go far beyond their use in cryptocurrencies [3–6]. Recently, blockchains have entered numerous domains of applications, such as supply chains, digital twins, insurance, healthcare, or energy. In brief, a blockchain is a decentralized, distributed, potentially public, and immutable log of objects.

Blockchains can be of different types. They can be public as Bitcoin or Ethereum, where the consensus protocol is executed between many pseudonymous participants. Here, the blockchain can be read and written by everyone. Such public blockchains can also be viewed as permissionless because everyone can join the system, participate in the consensus protocol, and establish smart contracts. Blockchains, however, can also be private (also called enterprise or permissioned blockchains) such as Hyperledger, Ethereum Enterprise, Ripple, or Quorum. Here, all the participants and their (digital) identities are known to one or more trusted organizations. Actors have write and read permissions. Such private blockchains can thus be viewed as permissioned because they restrict the actors who can contribute to the consensus on the system state to validate the block transactions. Once an object (such as a block or a transaction) is included in the blockchain (be it private or public), it is persisted and cannot be altered ever again. While immutability is a crucial property of the blockchain, it is often desirable to allow breaking the immutability for personal and legal reasons.

The debate about the immutability of the blockchain becomes more acute due to the adoption of the General Data Protection Regulation (GDPR) by the European Union

(EU). Several provisions of the GDPR are essentially incompatible with the immutable blockchains. In particular, the GDPR imposes that the data have the right to be forgotten, while blockchains such as Bitcoin and Ethereum do not allow to remove any data [7]. In addition, by using the immutability of a blockchain, malicious users can broadcast illegal or harmful data, such as (child) pornography and violence information around the world by spending a small fee. The data will be permanently stored and cannot be modified after they are stable on the chain. It is an enormous challenge for law enforcement agencies such as Interpol [8, 9]. One idea is to "filter" all incoming data to check for malicious content before inserting the data into the chain. However, the recent work of Matzutt et al. [10] showed that the above idea is not feasible. Hence, how to skillfully break the immutability of blockchains is an important and urgent problem to be solved.

To solve the above problem, Ateniese et al. [11] first introduced the concept of redactable blockchain and proposed an elegant solution based on chameleon hash functions [12]. The solution addresses the redaction problem of blockchains at the block level, which is coarse grained.

The redactable blockchain should meet the following two properties: (1) the originator of a transaction can specify a fine-grained access control policy about who can modify the transaction and which portions of the transaction can be redacted; (2) the harmful information contained in the previous block can be removed. Unfortunately, there is no redactable blockchain that meets both requirements.

In this paper, we explored how to effectively realize the fine-grained redactable blockchain. Our thought for realizing fine-grained redaction and effective deletion of harmful data simultaneously is shown in Figure 1. In order to support fine-grained access control, a promising way is to adopt the policy-based chameleon hash function (PCH) [13], which allows the originator of a transaction to specify a fine-grained access control policy about who can modify the transaction. However, it may incur the following issue by adopting the PCH. The malicious originator of the transaction may design an access policy that only allows him/her to modify the transaction to store undeletable harmful information in a blockchain. This does not satisfy the second property. To solve the above problem, we try to combine the technology proposed in [14]. The technology allows all users to create removal transactions by spending some transaction fees. Miners then vote on the transaction, and the harmful information is removed if enough votes are collected within a period of time. Obviously, this does not motivate users to actively remove harmful information from the chain because the user is not only rewarded for doing so but also needs to spend transaction fees. In order to motivate users, in this paper, the users create removal transactions without spending transaction fees. If the transaction passes the verification, the originator will obtain the reward paid by the malicious user who posted the harmful information. In addition, this technique only supports the deletion of additional information in the block and needs to store some "old state," that is, the hash value of the original transaction.

In practice, the redactable blockchains should meet the following three properties: (1) the originator of a transaction can specify a fine-grained access control policy about who can modify the transaction and which portions of the transaction can be redacted; (2) the harmful information contained in the previous block can be removed; (3) the data type that can be redacted is various. In order to support the redaction of various data types, we adopt the idea of the scheme in [15]. In this paper, the blockchain protocol not only supports removing additional information of the block but also redacting UTXO in the transaction. In order to reduce the storage space, we try to adopt a policy-based sanitizable signature [16]. However, in this way, the number of blocks of the signed data cannot be changed, and the set of inadmissible blocks needs to be stored. To solve this problem, we propose an improved policy-based sanitizable signature which allows that the number of blocks of the message m can be changed.

1.1. Contributions. In this paper, we first explore how to effectively realize the fine-grained redaction of blockchains while removing the harmful data. We then propose a fine-grained and controllably redactable blockchain protocol with harmful data forced removal. In a nutshell, the contribution of this paper can be summarized as follows:

(i) We propose a fine-grained and controllably redactable blockchain protocol with harmful data forced removal. Our scheme not only supports the usual redaction of transactions but also the forced removal of harmful information in the blockchain. The originator of the transaction can specify a fine-grained access control structure about who can redact the transaction and which portions of the transaction can be redacted. Authorized users may spend transaction fees to initiate a redaction transaction to redact the above transaction. Any user can initiate a transaction that contains the index of the block included harmful information without spending transaction fees. If the block does contain the harmful information, the miner who creates the new block can forcibly delete the harmful information. Thus, the harmful data can be removed; even the malicious users specify an access control that only they can modify the data. The user who provided the index of the block can receive the reward which is borne by the malicious user who initially posted the data. The malicious user will be blacklisted if the rewards are not paid within a period of time, and any transaction about the user will not be performed later. Furthermore, the scheme supports not only the redaction of additional data but also UTXO, i.e., unspent transaction outputs.

(ii) We present an improved policy-based sanitizable signature scheme, which is based on the scheme in [16]. In our scheme, the number of blocks of the signed data can be changed, and the set of

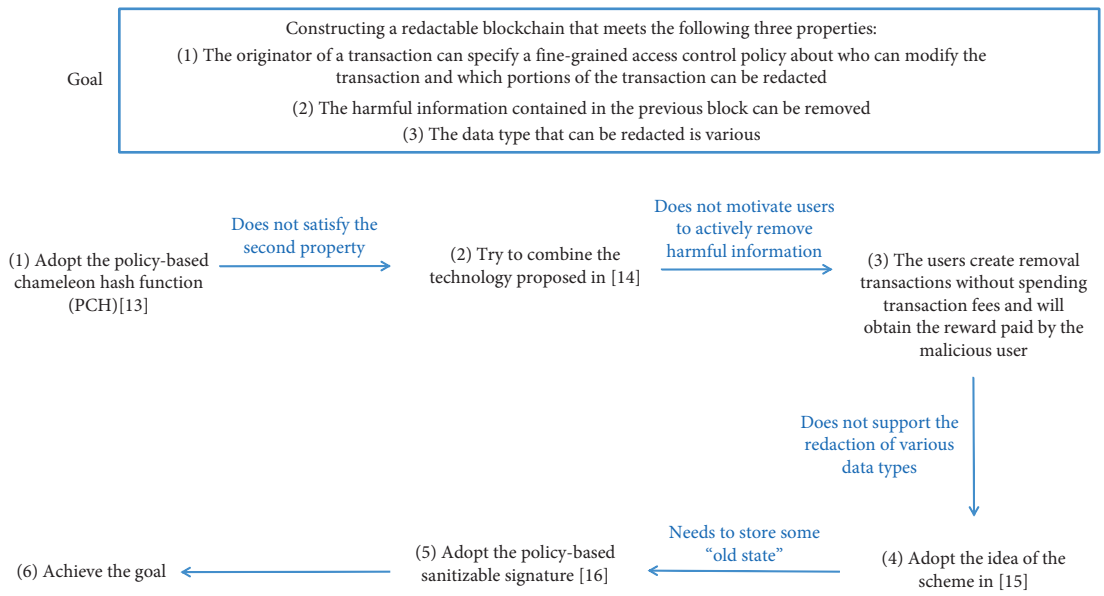

Figure 1: The flowchart of the idea.

inadmissible blocks does not need to be stored. Users who satisfy the access control policy can modify the portions of the signed data that are allowed to be modified. The authorized users can generate the valid signatures for the modified data without interacting with the original signer. The data owner does not need to collect the identities of the candidate authorized users in advance as the proxy signature schemes would require.

(iii) We demonstrate that the proposed scheme is secure and feasible via formal security analysis and proof-of-concept implementation. Specifically, we implement a full-fledged blockchain system, which achieves all the basic functionalities of Ethereum Enterprise. Separately, the blockchain system, including a subset of Ethereum Enterprise's script language, allows the authorized user to redact the transaction and the miner to delete the harmful data. We evaluate the performance of the blockchain system for chain validation in different scenarios. The results show that the redactable blockchain protocol produces only an insignificant (no more than 3.8%) overhead compared to the immutable blockchain.

1.2. Related Work. The concept of sanitizable signature was introduced by Ateniese et al. [17]. A sanitizable signature scheme allows a sanitizer to update the signed data without interacting with the original signer. In order to ensure the security of the scheme, two necessary security requirements are defined in their scheme: (1) unforgeability, that is, only authorized sanitizers can generate the new valid signatures for the updated data; (2) transparency, that is, the updated data and their signatures are indistinguishable from the original information and corresponding signatures. Unfortunately, they did not give a complete definition of the sanitizable signature nor did they provide the formal security analysis. Brzuska et al. [18, 19] provided the formal definition of sanitizable signatures and gave the formalized definition of the basic security requirements. They introduced five formal security requirements, unforgeability, immutability, privacy, transparency, and accountability, and analyzed the relationships between these security requirements. Canard et al. [20] proposed a generic construction of the trapdoor sanitizable signature. In this scheme, the sanitizer can generate the valid signature for the updated data after receiving the trapdoor key from the original signer. Using an accountable chameleon hash, Lai et al. [21] proposed an accountable trapdoor sanitizable signature. However, neither of the above two schemes gives the concrete construction of the sanitizable signature. After that, many concrete sanitizable signature schemes were proposed [22–24]. All of the above sanitizable schemes are not suitable for blockchain rewriting since none of the aforementioned schemes support fine-grained control over candidate sanitizers.

Attribute-based encryption schemes can provide fine-grained access control [25–27]. In order to provide fine-grained access control, some attribute-based sanitizable signature schemes are proposed [16, 28–30]. The scheme in [28] did not give the specific construct of the attribute-based sanitizable signature. The scheme in [29] did not support the expressive access structure. The scheme in [30] only provided an all-or-nothing solution for data modification. The number of blocks of the signed data cannot be changed, and the set of inadmissible blocks needs to be stored in [16]. In a real environment of blockchain rewriting, the number of blocks of the transaction may be changed, and the set of inadmissible blocks does not need to be contained in its signature. Therefore, in this paper, we improve the policy-based sanitizable signature scheme [16] and propose an

improved policy-based sanitizable signature. In this paper, the number of blocks of the signed data can be changed, and the set of inadmissible blocks does not need to be stored. Furthermore, we present a fine-grained and controllably redactable blockchain protocol with harmful data forced removal based on the improved policy-based sanitizable signature scheme.

1.3. Organization. The rest of this paper is organized as follows. In Section 2, we briefly review the preliminaries required in this paper. The system model and design goals are given in Section 3. In Section 4, we introduce the proposed improved policy-based sanitizable signature scheme. We describe the proposed blockchain protocol in Section 5. In Section 6, we introduce the security analysis of the proposed protocol. We evaluate the performance of the proposed protocol in Section 7. Finally, we come to the conclusion in Section 8.

2. Preliminaries

2.1. Notions. We list the notations used in our scheme in Table 1.

2.2. Access Structure. A collection $\mathbb{A} \in 2^{\mathbb{U}} \backslash \{\varnothing\}$ is an access structure on $\mathbb{U}$, where $\mathbb{U}$ denotes attributes' universe. If a set is contained in $\mathbb{A}$, it is the authorized set. Otherwise, it is an unauthorized set. A collection $\mathbb{A}$ is monotone if $C \in \mathbb{A}$ for $\forall B, C \in \mathbb{A}$ and $B \subseteq C$.

2.3. Public Key Encryption. A public key encryption scheme Π consists of the following five algorithms:

(i) $\mathrm{PPGen}_\Pi(1^\kappa)$: this algorithm takes the security parameter κ as the input and outputs the public parameters PP_Π.

(ii) $\mathrm{KGen}_\Pi(\mathrm{PP}_\Pi)$: this algorithm takes the public parameters PP_Π as the input and outputs the public and private key $(\mathrm{pk}_\Pi, \mathrm{sk}_\Pi)$.

(iii) $\mathrm{Enc}_\Pi(\mathrm{pk}_\Pi, m)$: this algorithm takes the public key pk_Π and the message m as the input and outputs a ciphertext c.

(iv) $\mathrm{Dec}_\Pi(\mathrm{sk}_\Pi, c)$: this algorithm takes the private key sk_Π and the ciphertext c as the input and outputs the message m.

(v) $\mathrm{KVrf}_\Pi(\mathrm{sk}_\Pi, \mathrm{pk}_\Pi)$: this algorithm takes the public and private key $(\mathrm{pk}_\Pi, \mathrm{sk}_\Pi)$ as the input and outputs 1 if sk_Π belongs to pk_Π. Otherwise, it outputs 0.

The detailed definition of correctness and security of the public key encryption (PKE) is given in [16]. In this paper, we require correctness and IND-CCA2 security for PKE.

Definition 1. (Π IND-CCA2 security). A public encryption scheme Π is IND-CCA2 secure [16] if for any probabilistic polynomial-time (PPT) adversary $\mathscr{A}$, there exists a negligible function ν such that

Table 1: Notations.

Notation	Meaning
$\mathbb{A}$	A monotone collection
$\mathbb{U}$	The attributes' universe
Π	A public key encryption scheme
k	The security parameter
PP_Π	The public parameters of Π
$(\mathrm{pk}_\Pi, \mathrm{sk}_\Pi)$	The public and private key of Π
m	The message
c	The ciphertext
Σ	A digital signature scheme
PP_Σ	The public parameters of Σ
$(\mathrm{pk}_\Sigma, \mathrm{sk}_\Sigma)$	The signer's public and private key in Σ
σ	The signature in Σ
L	A NP-language
Ω	A noninteractive proof system for L
crs_Ω	A common reference string
x	The statement
ω	The corresponding witness
π	The proof
$\mathrm{PP}_{\mathrm{PCH}}$	The public parameters of PCH
$(\mathrm{sk}_{\mathrm{PCH}}, \mathrm{pk}_{\mathrm{PCH}})$	The master key pair of PCH
$\mathbb{S}$	The set of attributes
$\mathrm{sk}_{\mathbb{S}}$	The user's secret key in PCH
h	The hash value
r	The randomness
m'	The modified message
r'	The new randomness
$\mathrm{PP}_{\mathrm{P3S}}$	The public parameters of P3S
$(\mathrm{sk}_{\mathrm{P3S}}, \mathrm{pk}_{\mathrm{P3S}})$	The master key pair of P3S
$(\mathrm{sk}^{\mathrm{sig}}_{\mathrm{P3S}}, \mathrm{pk}^{\mathrm{sig}}_{\mathrm{P3S}})$	The signer's key pair in P3S
$(\mathrm{sk}^{\mathrm{san}}_{\mathrm{P3S}}, \mathrm{pk}^{\mathrm{san}}_{\mathrm{P3S}})$	The sanitizer's key pair in P3S
M	The description of modification

$$\left| \Pr\left[\mathrm{Exp}^{\mathrm{IND-CCA2}}_{\mathscr{A},\Pi}(k) = 1\right] - \frac{1}{2} \right| \leq \nu(k). \tag{1}$$

The corresponding experiment is depicted in Figure 2.

2.4. Digital Signature. A digital signature scheme Σ consists of the following four algorithms:

(i) $\mathrm{PPGen}_\Sigma(1^\kappa)$: this algorithm takes the security parameter κ as the input and outputs the public parameters PP_Σ.

(ii) $\mathrm{KGen}_\Sigma(\mathrm{PP}_\Sigma)$: this algorithm takes the public parameters PP_Σ as the input and outputs signer's public and private key $(\mathrm{pk}_\Sigma, \mathrm{sk}_\Sigma)$.

(iii) $\mathrm{Sign}_\Sigma(\mathrm{sk}_\Sigma, m)$: this algorithm takes the private key sk_Σ and the message m as the input and outputs the signature σ.

(iv) $\mathrm{Verf}_\Sigma(\mathrm{pk}_\Sigma, m, \sigma)$: this algorithm takes the public key pk_Σ, the message m, and the signature σ as the input and outputs 1 if σ is valid. Otherwise, it outputs 0.

The formal security definition of the digital signature is given in [16]. In this paper, we require correctness and existential unforgeability (eUNF-CMA) for the digital signature.

$\mathrm{Exp}^{IND-CCA2}_{\mathcal{A},\Pi}(\mathrm{k})$
$\mathrm{PP}_\Pi \leftarrow_r PPGen_\Pi(1^k)$
$(sk_\Pi, pk_\Pi) \leftarrow_r KGen_\Pi(\mathrm{PP}_\Pi)$
$b \leftarrow_r \{0,1\}$
$((m_0^*, m_1^*), \mathrm{state}_\mathcal{A}) \leftarrow_r \mathcal{A}^{Dec_\Pi(sk_\Pi,\cdot)}(pk_\Pi)$
$If |m_0^*| \neq |m_1^*| \vee m_0^* \notin \mathcal{M} \vee m_1^* \notin \mathcal{M}$:
$c^* \leftarrow \perp$
$Else$:
$c^* \leftarrow_r Enc_\Pi(pk_\Pi, m_b^*)$
$b^* \leftarrow_r \mathcal{A}^{Dec'_\Pi(sk_\Pi,\cdot)}(state_\mathcal{A}, c^*)$
$where\ Dec'_\Pi$ on input sk_Π and c :
$return \perp$ if $c = c^*$
$return\ Dec_\Pi(sk_\Pi, c)$
$return\ 1$ if $b^* = b$
$return\ 0$

Figure 2: Π IND-CCA2 security.

Definition 2. (Σ unforgeability). A digital signature scheme Σ is unforgeable [16] if for any PPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\left|\Pr\left[\mathrm{Exp}^{\mathrm{eUNF-CMA}}_{\mathcal{A},\Sigma}(k) = 1\right]\right| \leq \nu(k). \tag{2}$$

The corresponding experiment is depicted in Figure 3.

2.5. Noninteractive Zero-Knowledge Proof (NIZK). Let $L = \{x | \exists \omega: R(x, \omega) = 1\}$, where L is a NP-language with associated witness relation R. A noninteractive proof system Ω for the language L consists of the following three algorithms:

(i) $\mathrm{PPGen}_\Omega(1^\kappa)$: this algorithm takes the security parameter κ as the input and outputs the common reference string (CRS) crs_Ω.

(ii) $\mathrm{Prove}_\Omega(\mathrm{crs}_\Omega, x, \omega)$: this algorithm takes CRS crs_Ω, the statement x, and the corresponding witness ω as the input and outputs the proof π.

(iii) $\mathrm{Verify}_\Omega(\mathrm{crs}_\Omega, x, \pi)$: this algorithm takes CRS crs_Ω, the statement x, and the proof π as the input and outputs 1 if π is valid. Otherwise, it outputs 0.

The security of the noninteractive zero-knowledge proof (NIZK) is given in [16]. In this paper, we require completeness for NIZK. In addition to completeness, we require two standard security notions for zero-knowledge proofs of knowledge: zero knowledge and simulation-sound extractability. We define them analogous to the definitions given in [16]. Informally speaking, zero knowledge says that the receiver of the proof π does not learn anything except the validity of the statement.

Definition 3. (completeness). A noninteractive proof system is called complete if for all $k \in N$, $\mathrm{crs}_\Omega \leftarrow_r \mathrm{PPGen}(1^k)$, $x \in L$, ω such that $R(x, \omega) = 1$, $\pi \leftarrow_r \mathrm{Prove}_\Omega(\mathrm{crs}_\Omega, x, \omega)$, it holds that $\mathrm{Verify}_\Omega(\mathrm{crs}_\Omega, x, \pi)$.

2.6. Policy-Based Chameleon Hashes. A policy-based chameleon hash (PCH) allows the user, who owns attributes' set that satisfied the access structure, to compute a hash collision [13]. Specifically, a PCH contains the following six PPT algorithms:

$\mathrm{Exp}^{eUNF-CMA}_{\mathcal{A},\Sigma}(\mathrm{k})$
$\mathrm{PP}_\Sigma \leftarrow_r PPGen_\Sigma(1^k)$
$(sk_\Sigma, pk_\Sigma) \leftarrow_r KGen_\Sigma(\mathrm{PP}_\Sigma)$
$Q \leftarrow \emptyset$
$(m^*, \sigma^*) \leftarrow_r A^{Sign'_\Sigma(sk_\Sigma,\cdot)}(pk_\Sigma)$
where $Sign'_\Sigma$ on input sk_Σ and m:
$\sigma \leftarrow_r Sign_\Sigma(sk_\Sigma, m)$
Set $Q \leftarrow Q \cup \{m\}$
return σ
return 1 if $Verf_\Sigma(pk_\Sigma, m^*, \sigma^*) = 1 \wedge m^* \notin Q$
return 0

Figure 3: Σ unforgeability.

(i) $\mathrm{PPGen}_{\mathrm{PCH}}(1^\kappa)$: this is the public parameters' generation algorithm. It takes the security parameter κ as the input and outputs the public parameters $\mathrm{PP}_{\mathrm{PCH}}$.

(ii) $\mathrm{MKeyGen}_{\mathrm{PCH}}(\mathrm{PP}_{\mathrm{PCH}})$: this is the master key generation algorithm. It takes the public parameter $\mathrm{PP}_{\mathrm{PCH}}$ as the input and outputs the master key pair $(\mathrm{sk}_{\mathrm{PCH}}, \mathrm{pk}_{\mathrm{PCH}})$.

(iii) $\mathrm{KGen}_{\mathrm{PCH}}(\mathrm{sk}_{\mathrm{PCH}}, \mathbb{S})$: this is the user's secret key generation algorithm. It takes the master secret key $\mathrm{sk}_{\mathrm{PCH}}$ and the set of attributes $\mathbb{S} \subseteq \mathbb{U}$ as the input and outputs the user's secret key $\mathrm{sk}_\mathbb{S}$.

(iv) $\mathrm{Hash}_{\mathrm{PCH}}(\mathrm{pk}_{\mathrm{PCH}}, \mathbb{A}, m)$: this is the hash algorithm. It takes the master public key $\mathrm{pk}_{\mathrm{PCH}}$, the access structure $\mathbb{A} \in 2^\mathbb{U} \backslash \{\varnothing\}$, and the message m as the input and outputs the hash value h and the randomness r.

(v) $\mathrm{Verify}_{\mathrm{PCH}}(\mathrm{pk}_{\mathrm{PCH}}, m, h, r)$: this is the verification algorithm. It takes the master public key $\mathrm{pk}_{\mathrm{PCH}}$, the message m, the hash value h, and the randomness r as the input and outputs a bit $b = 1$ if h and r are valid. Otherwise, $b = 0$.

(vi) $\mathrm{Adapt}_{\mathrm{PCH}}(\mathrm{pk}_{\mathrm{PCH}}, \mathrm{sk}_\mathbb{S}, m, m', h, r)$: this is the adaption algorithm. It takes the public key $\mathrm{pk}_{\mathrm{PCH}}$, the user's secret key $\mathrm{sk}_\mathbb{S}$, the message m, the modified message m', the hash value h, and some randomness r as the input and outputs a new randomness r'.

The detailed definition of correctness and security of the policy-based chameleon hash is given in [13].

2.7. Policy-Based Sanitizable Signature. A policy-based sanitizable signature (P3S) allows the user, who owns attributes' set that satisfied the access structure, to modify the data and generate the valid signatures for the modified data [16]. Specifically, a P3S contains the following ten PPT algorithms:

(i) $\mathrm{ParGen}_{\mathrm{P3S}}(1^\lambda)$: this is the public parameters' generation algorithm. It takes the security parameter λ as the input and outputs the public parameters $\mathrm{PP}_{\mathrm{P3S}}$.

(ii) $\mathrm{Setup}_{\mathrm{P3S}}(\mathrm{PP}_{\mathrm{P3S}})$: this is the master key generation algorithm. It takes the public parameters $\mathrm{PP}_{\mathrm{P3S}}$ as the input and outputs the master key pair $(\mathrm{sk}_{\mathrm{P3S}}, \mathrm{pk}_{\mathrm{P3S}})$.

(iii) $\mathrm{KGenSig}_{\mathrm{P3S}}(\mathrm{PP}_{\mathrm{P3S}})$: this is the signer's key pair generation algorithm. It takes the public parameters $\mathrm{PP}_{\mathrm{P3S}}$ as the input and outputs the signer's key pair $(\mathrm{sk}_{\mathrm{P3S}}^{\mathrm{sig}}, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{sig}})$.

(iv) $\mathrm{KGenSan}_{\mathrm{P3S}}(\mathrm{PP}_{\mathrm{P3S}})$: this is the sanitizer's key pair generation algorithm. It takes the public parameters $\mathrm{PP}_{\mathrm{P3S}}$ as the input and outputs the sanitizer's key pair $(\mathrm{sk}_{\mathrm{P3S}}^{\mathrm{san}}, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{san}})$.

(v) $\mathrm{Sign}_{\mathrm{P3S}}(\mathrm{PP}_{\mathrm{P3S}}, \mathrm{sk}_{\mathrm{P3S}}^{\mathrm{sig}}, m, A, \mathbb{A})$: this is the signing algorithm. It takes the public parameters $\mathrm{PP}_{\mathrm{P3S}}$, the signer's secret key $\mathrm{sk}_{\mathrm{P3S}}^{\mathrm{sig}}$, the message m, the description of admission A, and the access structure $\mathbb{A}$ as the input and outputs a signature σ.

(vi) $\mathrm{AddSan}_{\mathrm{P3S}}(\mathrm{sk}_{P3S}, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{san}}, \mathbb{S})$: this is the secret sanitizing key generation algorithm. It takes the master secret key $\mathrm{sk}_{\mathrm{P3S}}$, the sanitizer's public key $\mathrm{pk}_{\mathrm{P3S}}^{\mathrm{san}}$, and the set of attributes $\mathbb{S}$ as the input and outputs the secret sanitizing key $\mathrm{sk}_{\mathbb{S}}$ for the sanitizer.

(vii) $\mathrm{Verify}_{\mathrm{P3S}}(\mathrm{pk}_{\mathrm{P3S}}, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{sig}}, \sigma, m)$: this is the verification algorithm. It takes the master public key $\mathrm{pk}_{\mathrm{P3S}}$, the signer's public key $\mathrm{pk}_{\mathrm{P3S}}^{\mathrm{sig}}$, the signature σ, and the corresponding message m as the input and outputs a bit $b = 1$ if the signature σ is valid. Otherwise, $b = 0$.

(viii) $\mathrm{Sanitize}_{\mathrm{P3S}}(\mathrm{pk}_{\mathrm{P3S}}, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{sig}}, \mathrm{sk}_{\mathrm{P3S}}^{\mathrm{san}}, \mathrm{sk}_{\mathbb{S}}, \sigma, m, M)$: this is the new signature generation algorithm. It takes the master public key $\mathrm{pk}_{\mathrm{P3S}}$, the signer's public key $\mathrm{pk}_{\mathrm{P3S}}^{\mathrm{sig}}$, the sanitizer's secret key $\mathrm{sk}_{\mathrm{P3S}}^{\mathrm{san}}$, the secret sanitizing key $\mathrm{sk}_{\mathbb{S}}$, the signature σ, the corresponding message m, and the description of modification M as the input and outputs the new signature σ' for the modified message m'.

(ix) $\mathrm{Proof}_{\mathrm{P3S}}(\mathrm{pk}_{\mathrm{P3S}}, \mathrm{sk}_{\mathrm{P3S}}^{\mathrm{sig}}, \sigma, m)$: this is the proof generation algorithm. It takes the master public key $\mathrm{pk}_{\mathrm{P3S}}$, the signer's secret key $\mathrm{sk}_{\mathrm{P3S}}^{\mathrm{sig}}$, the signature σ, and the corresponding message m as the input and outputs the proof π_{P3S}.

(x) $\mathrm{Judge}_{\mathrm{P3S}}(\mathrm{PP}_{\mathrm{P3S}}, \mathrm{pk}_{\mathrm{P3S}}, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{sig}}, \sigma, m, \pi_{\mathrm{P3S}})$: this is the proof verification algorithm. It takes the public parameter $\mathrm{PP}_{\mathrm{P3S}}$, the master public key $\mathrm{pk}_{\mathrm{P3S}}$, the signer's public key $\mathrm{pk}_{\mathrm{P3S}}^{\mathrm{sig}}$, the signature σ, the corresponding message m, and the proof π_{P3S} as the input and outputs a bit $b = 1$ if the proof π_{P3S} is valid. Otherwise, $b = 0$.

The detailed definition of correctness and security of the policy-based sanitizable signature is given in [16].

2.8. Blockchain Protocol. Let Γ denote an immutable blockchain protocol such as Ethereum Enterprise. The nodes in the blockchain protocol obtain their local chain C based on a common genesis block. The nodes in the blockchain protocol collect transactions in the whole blockchain ecosystem and then package these transactions into a new block. The chain becomes longer as nodes agree on a new block. Nodes can access the blockchain protocol through the following interfaces.

(i) $\{C', \perp\} \leftarrow \Gamma \cdot \mathrm{updateChain}$: returns the chain C' if it is the longer and the valid chain in the blockchain ecosystem. Otherwise, it returns $\perp$.

(ii) $\{0, 1\} \leftarrow \Gamma \cdot \mathrm{validateChain}(C)$: takes the chain C as the input and outputs 1 iff the chain is valid according to the public set of rules.

(iii) $\{0, 1\} \leftarrow \Gamma \cdot \mathrm{validateBlock}(B)$: takes the block B as the input and outputs 1 iff the block is valid according to the public set of rules.

(iv) $\Gamma \cdot \mathrm{broadcast}(x)$: takes the transaction x as the input and broadcasts it to all nodes in the blockchain ecosystem.

3. Problem Formulation

3.1. System Model. As shown in Figure 4, the system model of the proposed redactable blockchain protocol consists of four entities: the trusted authority (TA), the miners, the users, and the authorized users. Note that the model in this paper is similar to the model in [13]. It is more applicable to permissioned blockchains, such as Hyperledger, Ethereum Enterprise, Ripple, and Quorum.

(i) Trusted authority (TA): trusted authority (TA) is fully honest and responsible for generating the signing private key for users who posted the redactable transaction, issuing the attributes and attributes' key for authorized users, and sending keys to miners.

(ii) Miners: miners are fully honest and have powerful computing resources. They are responsible for packaging transactions in the network to generate the new block and removing harmful information from the previous blocks.

(iii) Users: users may be malicious. They can post the usual transaction or the transaction containing the index of the block which includes harmful information to the network. Users get fine-grained control over which users can redact their usual transaction and which portions of the transaction can be redacted. The malicious users may specify an access structure that only allows themselves and their conspirators to redact the transaction.

(iv) Authorized users: the authorized users are semi-honest in the sense that they can modify the portions of the transaction that are allowed to be modified and generate the new valid signatures for the updated data that are indistinguishable from the signatures that the originator generated for the original transaction.

3.2. Design Goals. In order to realize a "healthy" blockchain protocol, the proposed fine-grained and controllably redactable blockchain with harmful data forced removal should satisfy the following properties:

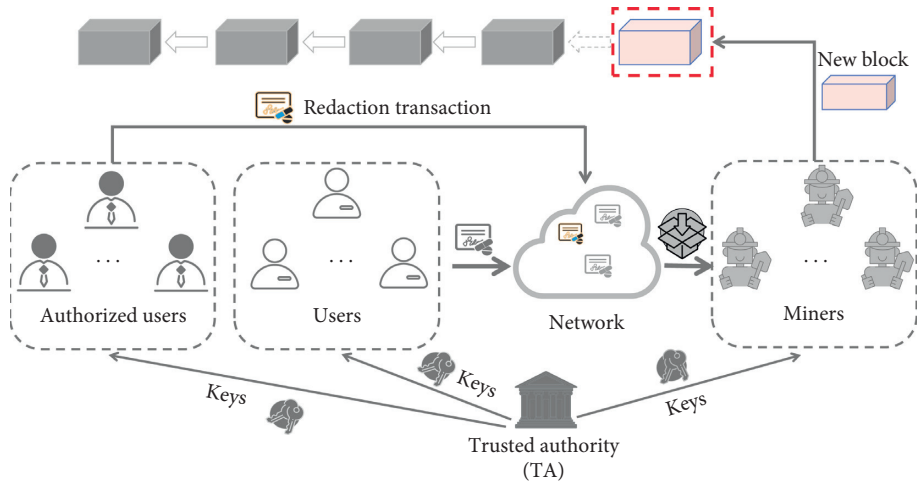

Figure 4: The system model.

(i) Controlled redaction: only authorized users can redact the portions of the transaction that are allowed to be redacted.

(ii) Accountability: the authorized user who redacts the transaction can be tracked.

(iii) Correctness: correctness ensures that the redacted blockchain is "healthy." Specifically, a "healthy" blockchain should meet the following characteristics:

(a) Chain growth: let C_1 and C_2 denote two chains possessed by two honest users at rounds r_1 and r_2, respectively. Then, $\text{len}(C_2) - \text{len}(C_1) \geq \tau \cdot (r_2 - r_1)$, where τ is the speed coefficient and $r_2 > r_1$.

(b) Chain quality: generally speaking, the chain quality says that the ratio of adversarial blocks in any segment of a chain held by an honest party is no more than a fraction $0 < \mu \leq 1$, where μ is the fraction of resources controlled by the adversary.

(c) Editable common prefix: the usual common prefix says that if C_1 and C_2 are two chains possessed by two honest users at rounds r_1 and r_2, for $r_2 > r_1$, C_1 is a prefix of C_2. It can be formally denoted as $C_1^k \leq C_2$, where C_1^k is the chain obtained by removing the last k blocks from C_1, $k \in \mathbb{N}$ is the common prefix parameter. Note that the proposed editable blockchain inherently does not satisfy the common prefix. Suppose the voting phase for the redaction transaction T_i^* is still on at round r_1. At round r_2, the voting phase is complete, and T_i^* replaces T_i, i.e., the redacted block B_i^* replaces B_i. In C_1^k, the i-th block is B_i instead of B_i^* as in C_2. Thus, C_1^k is not the common prefix of C_1 and C_2. We extend this definition. The chains C_1 and C_2 satisfy one of the following:

(1) $C_1^k \leq C_2$

(2) The voting phase is complete, and B_i^* replaces B_i if $B_i^* \in C_2^{(r_2 - r_1)+k}$, $B_i^* \notin C_1^k$

3.3. Threat Model

Definition 4. (controlled redaction). Controlled redaction ensures that only authorized users can redact the portions of the transaction that are allowed to be redacted. In order to formally describe the controlled redaction, we introduce a game between the challenger $\mathscr{C}$ and the adversary $\mathscr{A} = (\mathscr{A}_1, \mathscr{A}_2)$. Here, we consider two adversaries. One of the adversaries is the adversary $\mathscr{A}_1$, who does not possess the attributes' set which satisfies the access control policy. Another is the adversary $\mathscr{A}_2$, who tries to redact the inadmissible portions of the transaction. In order to show how $\mathscr{A}_1$ and $\mathscr{A}_2$ attack the redactable blockchain protocol, we introduce the game between the challenger $\mathscr{C}$ and adversaries $\mathscr{A}_1$ and $\mathscr{A}_2$, respectively.

Firstly, we describe the game between the challenger $\mathscr{C}$ and the adversary $\mathscr{A}_1$. Trusted authority (group manager) is viewed as a challenger $\mathscr{C}$, and the unauthorized user is viewed as an adversary $\mathscr{A}_1$. This game includes the following phases:

(i) Setup phase: the challenger $\mathscr{C}$ runs the $\text{ParGen}_{\text{P3S}}$ and $\text{Setup}_{\text{P3S}}$ algorithm to generate the public parameters PP_{P3S} and the master private/public key pair $(\text{sk}_{\text{P3S}}, \text{pk}_{\text{P3S}})$. Then, $\mathscr{C}$ holds the master private key sk_{P3S} locally. Finally, $\mathscr{C}$ sends the master public key pk_{P3S} and the public parameters PP_{P3S} to the adversary $\mathscr{A}_1$.

(ii) Query phase:

(a) $\text{KGenSan}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_1$ queries sanitizer's private/public key pair for the public parameters PP_{P3S}. $\mathscr{C}$ runs $\text{KGenSan}_{\text{P3S}}$ algorithm and returns the private/public key pair (x_2, y_2) to $\mathscr{A}_1$.

(b) Sign queries: the adversary $\mathscr{A}_1$ queries the signature for the master public key pk_{P3S}, the signature for the transaction m, the set of admissible blocks A, and the access structure $\mathbb{A}$. $\mathscr{C}$ runs $\text{KGenSig}_{\text{P3S}}$ to generate the signing key and then runs Sign algorithm to produce the

signature σ. Finally, $\mathscr{C}$ returns the signature σ to $\mathscr{A}_1$.

(c) $\text{AddSan}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_1$ queries the sanitizer's attribute key for sk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{San}}$, and the attributes' set $\mathbb{S}$ such that $\mathbb{A}(\mathbb{S}) = 0$. $\mathscr{C}$ runs $\text{AddSan}_{\text{P3S}}$ algorithm and returns the sanitizer's attribute key $\text{sk}_{\mathbb{S}} \leftarrow (\sigma_{\text{sk}_{\mathbb{S}}}, \text{sk}_{\mathbb{S}}')$ to $\mathscr{A}_1$.

(d) $\text{Verify}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_1$ queries the verification result for pk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{A}_1$.

(e) $\text{Sanitize}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_1$ queries the sanitizable signature for pk_{P3S}, pk_{P3S}^{Sig}, sk_{P3S}^{San}, $sk_{\mathbb{S}}$, m, σ, and m'. $\mathscr{C}$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm and returns the new signature σ' to $\mathscr{A}_1$.

(f) $\text{Proof}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_1$ queries $(\pi_{\text{P3S}}, \text{pk})$ for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Proof}_{\text{P3S}}$ algorithm and returns $(\pi_{\text{P3S}}, \text{pk})$ to $\mathscr{A}_1$.

(g) $\text{Judge}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_1$ queries the judge result for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Judge}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{A}_1$.

(iii) Challenge phase: the adversary $\mathscr{A}_1$ adaptively chooses the authorized user's attributes' set $\mathbb{S}$ $(\mathbb{A}(\mathbb{S}) = 0)$. Then, $\mathscr{A}_1$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm to generate the challenged signature σ^* for the challenged transaction m^*. Finally, the adversary $\mathscr{A}_1$ sends $(\mathbb{S}, m^*, \sigma^*)$ to $\mathscr{C}$.

(iv) Verify phase: the adversary $\mathscr{A}_1$ performs polynomial queries as in the query phase. Consider the adversary $\mathscr{A}_1$ has made L queries, and let $Q = \{\text{sk}_{\mathbb{S}}, \mathbb{S}, m_i, A_i, \mathbb{A}_i, \sigma_i\}_{i=1}^{[|Q|]}$ denote the set of information obtained through these queries. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}_{\text{P3S}}^{\text{Sig}}, A, \mathbb{A}, m^*, \sigma^*)$ algorithm and outputs a bit b_0. If $b_0 = 1$, $\mathscr{C}$ checks whether there exists an $i \in [|Q|], \sigma^*)$ such that $\mathbb{A}(\mathbb{S}) = 0$. If there is such an i, the challenger $\mathscr{C}$ outputs $b_1 = 1$. Otherwise, $\mathscr{C}$ outputs $b_1 = 0$.

We say that the adversary $\mathscr{A}_1$ wins if $b_1 = 1$. In the above game, we want to show that the adversary $\mathscr{A}_1$, who does not possess the attributes' set $\mathbb{S}$ such that $\mathbb{A}(\mathbb{S}) = 0$, should not generate the new valid witness for the transaction. The adversary's goal is to correctly generate the valid signature σ' for the transaction m^*. We set the advantage of a polynomial-time adversary $\mathscr{A}_1$ in this game to be $\Pr[b_1 = 1]$. We say the proposed scheme satisfies the unforgeability of the signature if for any polynomial-time adversary $\mathscr{A}_1$, $\Pr[b_1 = 1] < (1/\text{poly}(n))$ for sufficiently large n, where poly stands for a polynomial function.

Then, we describe the game between the challenger $\mathscr{C}$ and the adversary $\mathscr{A}_2$. Trusted authority (group manager) is viewed as a challenger $\mathscr{C}$, and the authorized user is viewed as an adversary $\mathscr{A}_2$. This game includes the following phases:

(i) Setup phase: the challenger $\mathscr{C}$ runs the $\text{ParGen}_{\text{P3S}}$ and $\text{Setup}_{\text{P3S}}$ algorithm to generate the public parameters PP_{P3S} and the master private/public key pair $(\text{sk}_{\text{P3S}}, \text{pk}_{\text{P3S}})$. Then, $\mathscr{C}$ holds the master private key sk_{P3S} locally. Finally, $\mathscr{C}$ sends the master public key pk_{P3S} and the public parameters PP_{P3S} to the adversary $\mathscr{A}_2$.

(ii) Query phase:

(a) $\text{KGenSan}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_2$ queries sanitizer's private/public key pair for the public parameters PP_{P3S}. $\mathscr{C}$ runs $\text{KGenSan}_{\text{P3S}}$ algorithm and returns the private/public key pair (x_2, y_2) to $\mathscr{A}_2$.

(b) Sign queries: the adversary $\mathscr{A}_2$ queries the signature for the master public key pk_{P3S}, the signature for the message m, the set of admissible blocks F, and the access structure $\mathbb{A}$. $\mathscr{C}$ runs $\text{KGenSig}_{\text{P3S}}$ to generate the signing key and then runs Sign algorithm to produce the signature σ. Finally, $\mathscr{C}$ returns the signature σ to $\mathscr{A}_2$.

(c) $\text{AddSan}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_2$ queries the sanitizer's attribute key for sk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{San}}$, and the attributes' set $\mathbb{S}$ such that $\mathbb{A}(\mathbb{S}) = 1$. $\mathscr{C}$ runs $\text{AddSan}_{\text{P3S}}$ algorithm and returns the sanitizer's attribute key $\text{sk}_{\mathbb{S}} \leftarrow (\sigma_{\text{sk}_{\mathbb{S}}}, \text{sk}_{\mathbb{S}}')$ to $\mathscr{A}_2$.

(d) $\text{Verify}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_2$ queries the verification result for pk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{A}_2$.

(e) $\text{Sanitize}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_2$ queries the sanitizable signature for pk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{Sig}}$, $\text{sk}_{\text{P3S}}^{\text{San}}$, $\text{sk}_{\mathbb{S}}$, m, σ, and m'. $\mathscr{C}$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm and returns the new signature σ' to $\mathscr{A}_2$.

(f) $\text{Proof}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_2$ queries $(\pi_{\text{P3S}}, \text{pk})$ for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Proof}_{\text{P3S}}$ algorithm and returns $(\pi_{\text{P3S}}, \text{pk})$ to $\mathscr{A}_2$.

(g) $\text{Judge}_{\text{P3S}}$ queries: the adversary $\mathscr{A}_2$ queries the judge result for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Judge}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{A}_2$.

(iii) Challenge phase: the adversary $\mathscr{A}_2$ adaptively chooses the authorized user's attributes' set $\mathbb{S}$ $(\mathbb{A}(\mathbb{S}) = 1)$. Then, $\mathscr{A}_2$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm to generate the challenged signature σ^* for the challenged message m^* which does not contain all inadmissible blocks. Finally, the adversary $\mathscr{A}_2$ sends $(\mathbb{S}, m^*, \sigma^*)$ to $\mathscr{C}$.

(iv) Verify phase: the adversary $\mathscr{A}_2$ performs polynomial queries as in the query phase. Consider the adversary $\mathscr{A}_2$ has made L queries, and let $Q = \{\text{sk}_{\mathbb{S}}, \mathbb{S}, m_i, A_i, \mathbb{A}_i, \sigma_i\}_{i=1}^{[|Q|]}$ denote the set of information obtained through these queries. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}_{\text{P3S}}^{\text{Sig}}, A, \mathbb{A}, m^*, \sigma^*)$ algorithm and outputs a bit b_0. If $b_0 = 1$, $\mathscr{C}$ checks whether there exists an $i \in [|Q|], m^*$ which does not contain all inadmissible blocks. If there is such an i, the challenger $\mathscr{C}$ outputs $b_1 = 1$. Otherwise, $\mathscr{C}$ outputs $b_1 = 0$.

We say that the adversary $\mathscr{A}_2$ wins if $b_1 = 1$. In the above game, we want to show that the adversary $\mathscr{A}_2$, who redacts

the inadmissible blocks, should not generate the new valid signature. The adversary's goal is to correctly generate the valid signature σ' for the message m^*. We set the advantage of a polynomial-time adversary $\mathscr{A}_2$ in this game to be $\Pr[b_1 = 1]$. We say the proposed scheme satisfies controlled redaction if for any polynomial-time adversary $\mathscr{A}_2$, $\Pr[b_1 = 1] < (1/\text{poly}(n))$ for sufficiently large n, where poly stands for a polynomial function.

Definition 5. (accountability). We say that the proposed fine-grained and controllably redactable blockchain with harmful data forced removal satisfies accountability if TA can extract signer's identity from any valid transaction's signature with nonnegligible probability.

4. The Improved Policy-Based Sanitizable Signature

4.1. Algorithm Definition. Let PCH denote a policy-based chameleon hash, Ω label a simulation-sound extractable noninteractive zero-knowledge proof (NIZK) system, f be a one-way function, Π denote an IND-CCA2-secure public key encryption scheme, and Σ be an eUNF-CMA-secure signature scheme. Specifically, the improved policy-based sanitizable signature is described as follows:

(i) $\text{ParGen}_{\text{P3S}}(1^\kappa)$: it takes a security parameter κ as the input and outputs $\text{PP}_{\text{P3S}} = (\text{crs}_\Omega, \text{PP}_\Pi, \text{PP}_\Sigma, \text{PP}_{\text{PCH}}, f, h)$, where $\text{PP}_\Pi \leftarrow \text{PPGen}_\Pi(1^\kappa)$, $\text{crs}_\Omega \leftarrow \text{PPGen}_\Omega(1^\kappa)$, $\text{PP}_\Sigma \leftarrow \text{PPGen}_\Sigma(1^\kappa)$, $\text{PP}_{\text{PCH}} \leftarrow \text{PPGen}_{\text{PCH}}(1^\kappa)$, $f: D_f \longrightarrow R_f$ is a one-way function, and H is a cryptographic hash function.

(ii) $\text{Setup}_{\text{P3S}}(\text{PP}_{\text{P3S}})$: it takes PP_{P3S} as the input and outputs $(\text{sk}_{\text{P3S}}, \text{pk}_{\text{P3S}}) \leftarrow (\text{sk}_{\text{PCH}}, \text{sk}_\Sigma), (\text{pk}_{\text{PCH}}, \text{pk}_\Sigma)$, where $(\text{sk}_{\text{PCH}}, \text{pk}_{\text{PCH}}) \leftarrow \text{MKeyGen}_{\text{PCH}}(\text{PP}_{\text{PCH}})$ and $(\text{sk}_\Sigma, \text{pk}_\Sigma) \leftarrow \text{KGen}_\Sigma(\text{PP}_\Sigma)$.

(iii) $\text{KGenSig}_{\text{P3S}}(\text{PP}_{\text{P3S}})$: it takes PP_{P3S} as the input and outputs $(\text{sk}^{\text{Sig}}_{\text{P3S}}, \text{pk}^{\text{Sig}}_{\text{P3S}}) \leftarrow ((x_1, \text{sk}'_\Sigma, \text{sk}_\Pi), (y_1, \text{pk}'_\Sigma, \text{pk}_\Pi))$, where $x_1 \leftarrow D_f$, $y_1 \leftarrow f(x_1)$, $(\text{sk}_\Pi, \text{pk}_\Pi) \leftarrow \text{KGen}_\Pi(\text{PP}_\Pi)$, and $(\text{sk}'_\Sigma, \text{pk}'_\Sigma) \leftarrow \text{KGen}_\Sigma(\text{PP}_\Sigma)$.

(iv) $\text{KGenSan}_{\text{P3S}}(\text{PP}_{\text{P3S}})$: it takes PP_{P3S} as the input and outputs (x_2, y_2), where $x_2 \leftarrow D_f$ and $y_2 \leftarrow f(x_2)$.

(v) $\text{Sign}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{sk}^{\text{Sig}}_{\text{P3S}}, m, A, \mathbb{A})$: it takes pk_{P3S}, $\text{sk}^{\text{Sig}}_{\text{P3S}}$, the message m, the set of admissible blocks A, and the access structure $\mathbb{A}$ as the input and outputs $\perp$ if $\mathbb{A} = \varnothing$. Otherwise, it outputs $\sigma \leftarrow (h, r, A, \sigma_m, \mathbb{A}, \pi, c)$, where $(h, r) \leftarrow \text{Hash}_{\text{PCH}}(\text{pk}_{\text{PCH}}, m, \mathbb{A})$, $\sigma_m \leftarrow \text{Sign}_\Sigma(\text{sk}'_\Sigma, (\text{pk}_{\text{P3S}}, \text{pk}^{\text{Sig}}_{\text{P3S}}, A, H(i \| m_{!A}), h, \mathbb{A}))$, $c \leftarrow \text{Enc}_\Pi(\text{pk}_\Pi, y_1)$, and $\pi \leftarrow \text{Prove}_\Omega\{(x_1, x_2, \text{sk}_\Pi, \sigma_{\text{sk}_\mathbb{S}})$: $(y_1 = f(x_1) \wedge c = \text{Enc}_\Pi(\text{pk}_\Pi, y_1) \wedge \text{KVrf}_\Pi(\text{sk}_\Pi, \text{pk}_\Pi) = 1) \vee (y_2 = f(x_2) \wedge c = \text{Enc}_\Pi(\text{pk}_\Pi, y_2) \wedge \text{Verf}_\Sigma(\text{pk}_\Sigma, (y_2, \text{pk}_{\text{P3S}}), \sigma_{\text{sk}_\mathbb{S}}) = 1)\}(l)$. Note that $l = (\text{PP}_{\text{P3S}}, \text{pk}_{\text{P3S}}, \text{pk}^{\text{Sig}}_{\text{P3S}}, h, r, m, A, \mathbb{A}, H(i \| m_{!A}), \sigma_m, c)$.

(vi) $\text{AddSan}_{\text{P3S}}(\text{sk}_{\text{P3S}}, \text{pk}^{\text{San}}_{\text{P3S}}, \mathbb{S})$: it takes sk_{P3S}, $\text{pk}^{\text{San}}_{\text{P3S}}$, and the attributes' set $\mathbb{S}$ as the input and outputs the sanitizing key $\text{sk}_\mathbb{S} \leftarrow (\sigma_{\text{sk}_\mathbb{S}}, \text{sk}'_\mathbb{S})$, where $\sigma_{\text{sk}_\mathbb{S}} \leftarrow \text{Sign}_\Sigma(\text{sk}_\Sigma, (\text{pk}^{\text{San}}_{\text{P3S}}, \text{pk}_{\text{P3S}}))$ and $\text{sk}'_\mathbb{S} \leftarrow \text{KGen}_{\text{PCH}}(\text{sk}_{\text{PCH}}, \mathbb{S})$.

(vii) $\text{Verify}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}^{\text{Sig}}_{\text{P3S}}, \sigma, m)$: it takes pk_{P3S}, $\text{pk}^{\text{Sig}}_{\text{P3S}}$, σ, and m as the input and outputs 1 if π and σ_m are valid, $\text{Verify}_{\text{PCH}}(\text{pk}_{\text{PCH}}, m, r, h) = 1$, and $H(i \| m_{!A})$ can be computed from the message m. Otherwise, it outputs $\perp$.

(viii) $\text{Sanitize}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}^{\text{Sig}}_{\text{P3S}}, \text{sk}^{\text{San}}_{\text{P3S}}, \text{sk}_\mathbb{S}, m, \sigma, m')$: it takes pk_{P3S}, $\text{pk}^{\text{Sig}}_{\text{P3S}}$, $\text{sk}^{\text{San}}_{\text{P3S}}$, $\text{sk}_\mathbb{S}$, m, σ, and m' as the input. If $\sigma_{\text{sk}_\mathbb{S}}$ or σ is not valid, it outputs $\perp$. Otherwise, the sanitizer computes $r' \leftarrow \text{Adapt}_{\text{PCH}}(\text{pk}_{\text{PCH}}, \text{sk}_\mathbb{S}, m, m', h, r)$, $c' \leftarrow \text{Enc}_\Pi(\text{pk}_\Pi, y_2)$, and $\pi' \leftarrow \text{Prove}_\Omega\{(x_1, x_2, \text{sk}_\Pi, \sigma_{\text{sk}_\mathbb{S}})$: $(y_1 = f(x_1) \wedge c' = \text{Enc}_\Pi(\text{pk}_\Pi, y_1) \wedge \text{KVrf}_\Pi(\text{sk}_\Pi, \text{pk}_\Pi) = 1) \vee (y_2 = f(x_2) \wedge c' = \text{Enc}_\Pi(\text{pk}_\Pi, y_2) \wedge \text{Verf}_\Sigma(\text{pk}_\Sigma, (y_2, \text{pk}_{\text{P3S}}), \sigma_{\text{sk}_\mathbb{S}}) = 1)\}(l)$. Note that $l = (\text{PP}_{\text{P3S}}, \text{pk}_{\text{P3S}}, \text{pk}^{\text{Sig}}_{\text{P3S}}, h, r', m', A, \mathbb{A}, H(i \| m_{!A}), \sigma_m, c')$. Then, the sanitizer sets $(\sigma', m') \leftarrow ((h, r', A, \sigma_m, \mathbb{A}, \pi, c'), m')$. If (σ', m') is not valid, this algorithm outputs $\perp$. Otherwise, it outputs (σ', m').

(ix) $\text{Proof}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{sk}^{\text{Sig}}_{\text{P3S}}, \sigma, m)$: it takes pk_{P3S}, $\text{sk}^{\text{Sig}}_{\text{P3S}}$, σ, and m as the input and outputs (π_{P3S}, pk), where $\text{pk} \leftarrow \text{Dec}_\Pi(\text{sk}_\Pi, c)$, $\pi_{\text{P3S}} \leftarrow \text{Prove}_\Omega\{(\text{sk}_\Pi, x_1)$: $\text{pk} = \text{Dec}_\Pi(\text{sk}_\Pi, c) \wedge \text{KVrf}_\Pi(\text{sk}_\Pi, \text{pk}_\Pi) = 1 \wedge y_1 = f(x_1)\}(l)$, and $l = (\text{PP}_{\text{P3S}}, \text{pk}_{\text{P3S}}, \text{pk}^{\text{ig}}_{\text{P3S}}, \sigma, \text{pk}, m)$.

(x) $\text{Judge}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}^{\text{Sig}}_{\text{P3S}}, \text{pk}, \pi_{\text{P3S}}, \sigma, m)$: it takes pk_{P3S}, $\text{pk}^{\text{Sig}}_{\text{P3S}}$, pk, π_{P3S}, σ, and m as the input. If σ and π_{P3S} are valid, it outputs 1. Otherwise, it outputs 0.

The improved policy-based sanitizable signature replaces the inadmissible block set $m_{!A}$ in [16] with $H(i \| m_{!A})$ to allow that the number of blocks of the message m can be changed, and the set of inadmissible blocks does not need to be stored. Here, $m_{!A}$ denotes the set of blocks that are not allowed to be modified. The security definition and analysis are given in Appendixes A and B, respectively.

5. The Proposed Protocol

5.1. An Overview. The workflow of the proposed blockchain protocol can be described as follows. Firstly, users can generate a local chain C based on the common genesis block genesis and initialize the redaction transaction list R, the removal transaction list D, the penalty payment transaction list P, and the blacklist L to be empty. After that, users run $\Gamma \cdot \text{updateChain}$ to obtain the longest chain in the blockchain network. When the user wants to redact the previous transaction, he/she first broadcasts a redaction transaction by spending some transaction fees. The transaction will be added to the list R if it is valid. The miners vote on the transaction. The transaction can be executed if enough votes are collected within a period of time as shown in Figure 5. When a user finds harmful information contained in a block, he/she creates a removal transaction containing the index of the block without spending transaction fees. Miners create

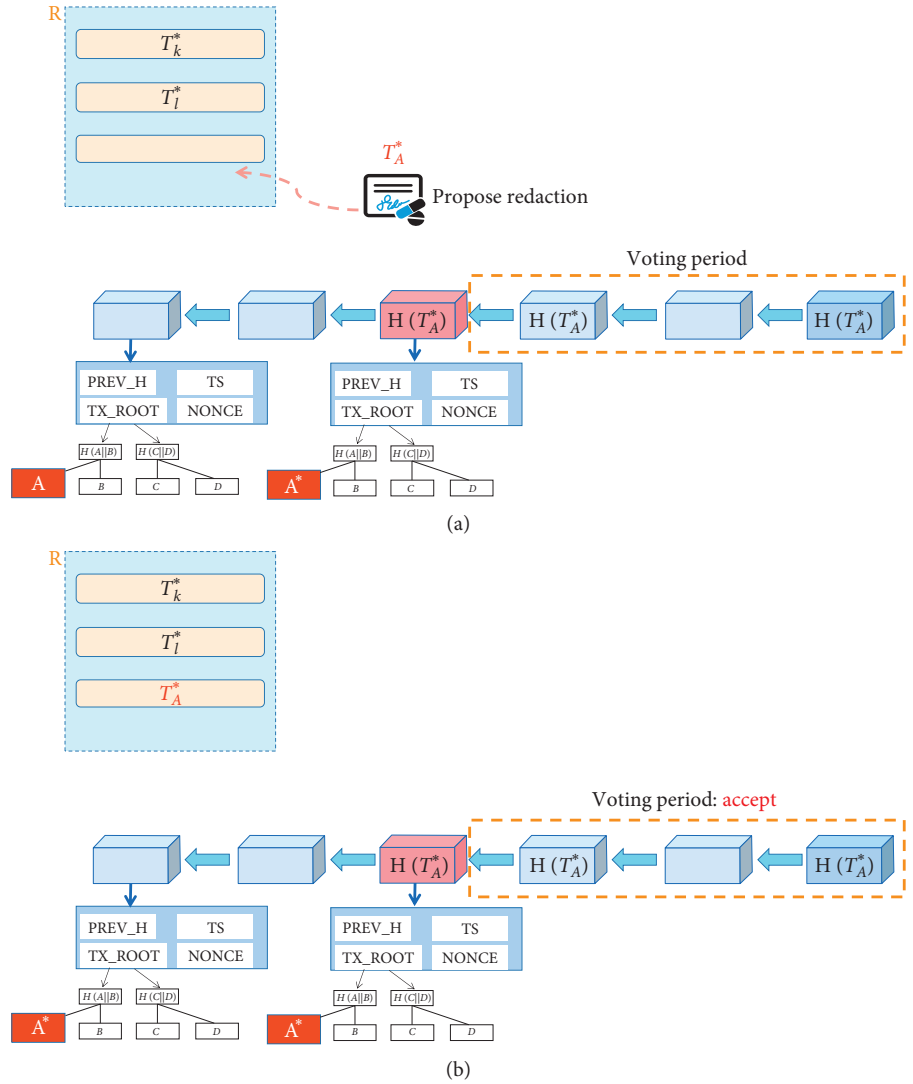

Figure 5: The redaction of the transaction. (a) Proposing a redaction A^* for a transaction A. (b) After a successful voting phase, A^* replaces A in the chain.

new blocks that contain at least one transaction in D and one in P if they are not empty. The miner removes the harmful information from the block according to the provided index. Meanwhile, the miner generates a penalty payment transaction added to P as shown in Figure 6. The transaction will be removed from list P after the penalty is paid by the malicious user. If the malicious user fails to pay the penalty within a period of time, he/she will be added to the blacklist L. After that, all transactions relating to the malicious user will never be performed.

5.2. Description of the Proposed Protocol. The proposed blockchain protocol runs in a sequence of rounds r and consists of the following six algorithms (Figures 7–10):

(i) Initialization: get the local chain $C \leftarrow$genesis, where genesis denotes a common genesis block. Set round $r \leftarrow 1$, and initialize empty lists R, D, P, and L.

(ii) Chain update: at the beginning of each round r, users run $\{C', \perp\} \leftarrow \Gamma \cdot \text{updateChain}$ to get the longest chain C' in the blockchain network.

(iii) Propose a redaction: the user proposes a redaction of the transaction T_A by spending some transaction fees.

  (a) Firstly, the user creates a redaction transaction redact T using the new transaction T_A^* as shown in Figure 7. In this process, the improved policy sanitizable signature is used to generate the witness for the transaction. We can see from Figure 7 that the hash values h for T_A and (T_A^*) are the same. Therefore, the hash value of this block will not be changed after redacting the transaction.

  (b) Then, he/she runs $\Gamma \cdot \text{broadcast}(T_A^*)$ to broadcast the redacted transaction to the blockchain network.

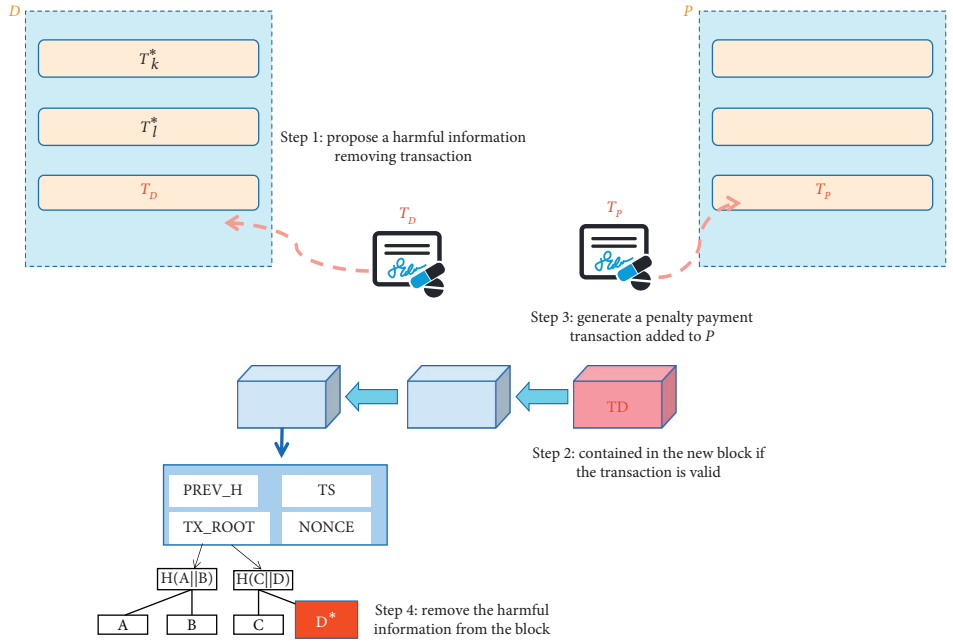

Figure 6: The removal of harmful information.

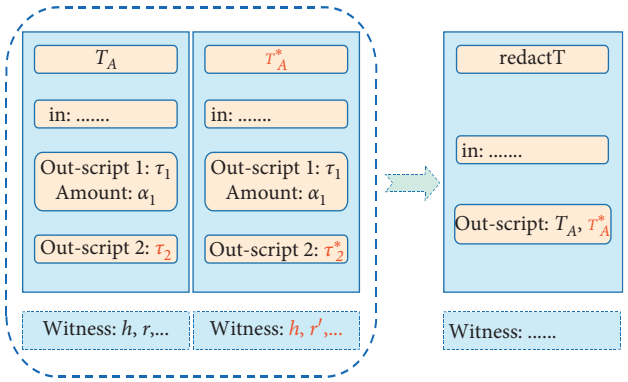

Figure 7: The transaction redact T.

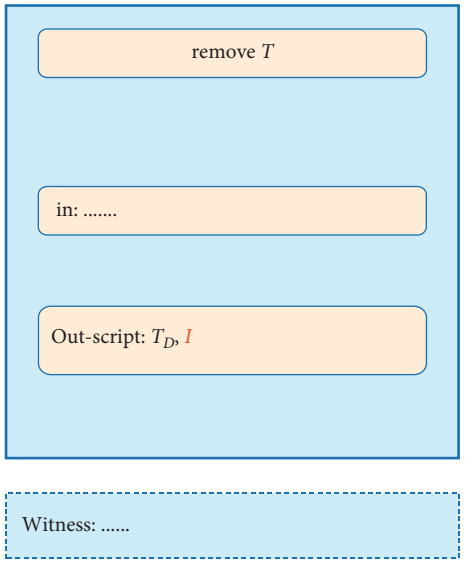

Figure 8: The transaction remove T.

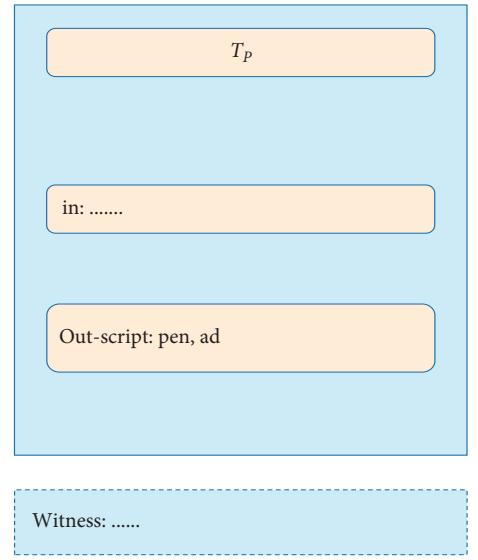

Figure 9: The transaction penatly T.

(c) Finally, miners add the transaction rdact T to the list R if the data τ_2 are UTXO. Otherwise, the transaction is discarded.

(iv) Propose a removal of harmful information when the user finds that the transaction T_D, contained in the block with the index I, has the harmful information.

(a) Firstly, as shown in Figure 8, the user creates a removal transaction remove T, which does not cost transaction fee and contains the block's index I and the transaction T_D.

(b) Then, he/she broadcasts the transaction remove T.

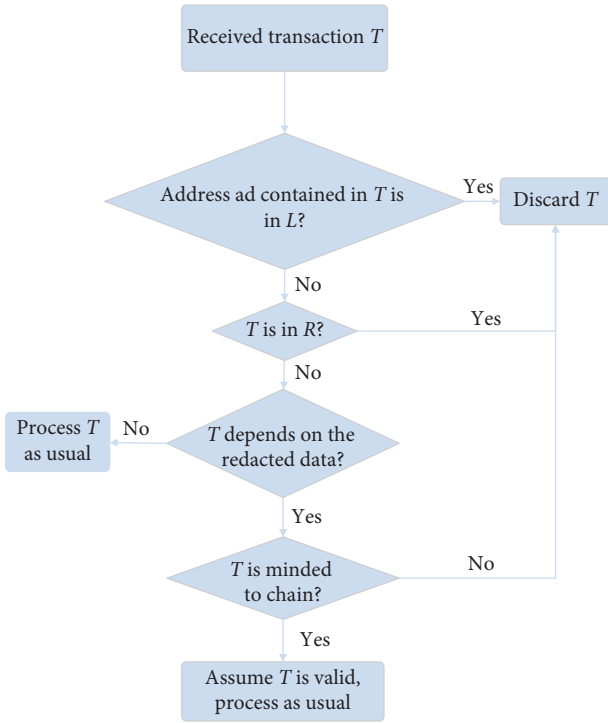

Figure 10: The verification of the received transaction T.

(c) The transaction remove T will be added to the list D if the block I does contain the harmful information. Meanwhile, the penalty payment transaction T_P will be created and added to the list P. As shown in Figure 9, the transaction T_P contains the amount of the penalty pen and the address ad of the malicious user who posts the harmful information. The transaction T_P will be removed from the list P after the malicious user pays the penalty.

(v) Redacting the chain:

(a) For the candidate transaction T_A in the list R, the miner substitutes it with the new transaction T_A^* if the voting process on it has been completed and enough votes $v \geq \rho$ have been collected within a period of time t. The transaction T_A is discarded if the votes $v < \rho$ within a period of time t. If the voting on T_A is still in progress, nothing will be done. Here, ρ denotes the threshold of votes and can be specified by consensus among all users in the blockchain network.

(b) For the candidate transaction T_D in the list D, the miner removes the harmful information from T_D which is contained in the block with the index I.

(c) For the candidate transaction T_P in the list P, the miner first verifies whether the malicious user pays the penalty within a period of time t_1. If the malicious user pays the penalty, the transaction is removed from P. If the malicious user does not pay the penalty within a specified period of time, the user is added to the blacklist L, and the transaction T_P is removed from the list P.

(vi) Creating a new block: the miner collects all transactions from the network for the $r-$th round and builds a new block B which meets the following conditions:

(a) It contains at least one transaction in D and one in P if they are not empty.

(b) It contains a vote $H(T_A)$ on the candidate transaction in the list R if the voting on T_A is still in process and the miner is willing to endorse.

(c) All transactions contained in it comply with the usual transaction rules in the Ethereum Enterprise blockchain, and the validation process is shown in Figure 10.

Finally, if all blocks contained in the local chain C satisfy $\Gamma \cdot \text{validateBlock}(B) = 1$ and $\Gamma \cdot \text{validateChain}(C) = 1$, the miner extends the local chain $C \leftarrow C \| B$ and broadcasts the extended chain to the blockchain network.

6. Security Analysis

In this section, we analyze the security of the fine-grained and controllably redactable blockchain protocol with harmful data forced removal in terms of correctness, controlled redaction, and accountability.

Theorem 1 (correctness). The correctness of a blockchain consists of the following three aspects:

(1) Chain growth: if the based immutable blockchain protocol Γ satisfies chain growth, the extended editable blockchain protocol Γ' also satisfies chain growth.

(2) Chain quality: if the based immutable blockchain protocol Γ satisfies chain quality, the extended editable blockchain protocol Γ' also satisfies chain growth for any $\rho > \mu$. Here, ρ denotes the ratio of blocks containing the votes of the redacted transaction within a period of time.

(3) Common prefix: if the based immutable blockchain protocol Γ satisfies the common prefix, the extended editable blockchain protocol Γ' also satisfies the common prefix.

Proof.

(1) Chain growth: we note that the redaction in Γ' cannot reduce the length of the chain C by removing a block from the chain. Thus, the redact operations have no effect on the length of the chain. In conclusion, Γ' satisfies chain growth if Γ satisfies chain growth.

(2) Chain quality: suppose the adversary $\mathcal{A}$ posts a malicious redaction transaction T_i^* for the previous

transaction T_i. $\mathscr{A}$ mines at most μ ratio of blocks in the voting phase because the adversary only has μ computational power. Thus, T_i^* cannot be performed due to $\rho > \mu$. In conclusion, only the honest redaction transaction T_i^* can be performed and added to the chain.

(3) Common prefix: if the chain C_2 is not redacted, Γ' runs as the immutable blockchain Γ. Thus, Γ' satisfies the common prefix. If the chain C_2 is redacted and the redacted block B_i^* replaces B_i in C_2, the voting phase for the block B_i^* is completed, and enough votes are received. In conclusion, the extended editable blockchain protocol Γ' also satisfies the common prefix. □

Theorem 2 (controlled redaction). In the proposed scheme, for each PPT adversary $\mathscr{A}$, it is computationally infeasible to generate a valid signature for the redacted transaction.

Proof. To prove this theorem, we consider two types of adversaries. One of the adversaries is the adversary $\mathscr{A}_1$ who does not possess the attributes' set which satisfies the access control policy. Another is the adversary $\mathscr{A}_2$, who tries to redact the inadmissible portions of the transaction. In order to show how $\mathscr{A}_1$ and $\mathscr{A}_2$ attack the redactable blockchain protocol, we introduce the two games between the challenger $\mathscr{C}$ and adversaries $\mathscr{A}_1$ and $\mathscr{A}_2$, respectively. Firstly, we define a game between the challenger $\mathscr{C}$ and the adversary $\mathscr{A}_1$.

Game 1: in Game 1, both the challenger $\mathscr{C}$ and the adversary $\mathscr{A}_1$ perform as defined in the security definition.

(i) Setup phase: the adversary $\mathscr{A}_1$ does as in the "Threat Model."

(ii) Query phase: the adversary $\mathscr{A}_1$ does as in the "Threat Model."

(iii) Challenge phase: the adversary $\mathscr{A}_1$ adaptively chooses the authorized user's attributes' set $\mathbb{S}$ ($\mathbb{A}(\mathbb{S}) = 0$). Then, $\mathscr{A}_1$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm to generate the challenged signature σ^* for the challenged transaction m^*. Finally, the adversary $\mathscr{A}_1$ sends $(\mathbb{S}, m^*, \sigma^*)$ to $\mathscr{C}$.

(iv) Verify phase: the adversary $\mathscr{A}_1$ performs polynomial queries as in the query phase. Consider the adversary $\mathscr{A}_1$ has made L queries, and let $Q = \{\text{sk}_{\mathbb{S}}, \mathbb{S}, m_i, A_i, \mathbb{A}_i, \sigma_i\}_{i=1}^{[|Q|]}$ denote the set of information obtained through these queries. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}_{\text{P3S}}^{\text{Sig}}, A, \mathbb{A}, m^*, \sigma^*)$ algorithm and outputs a bit b_0. If $b_0 = 1$, $\mathscr{C}$ checks whether there exists an $i \in [|Q|], \sigma^*)$ such that $\mathbb{A}(\mathbb{S}) = 0$. If there is such an i, the challenger $\mathscr{C}$ outputs $b_1 = 1$. Otherwise, $\mathscr{C}$ outputs $b_1 = 0$.

Suppose $b_1 = 1$, that is, the adversary $\mathscr{A}_1$ wins, we can say that the adversary $\mathscr{A}_1$ breaks the security of the policy-based sanitizable signature because the adversary's goal is to correctly generate the valid signature σ' for the transaction m^*. According to the security of the policy-based sanitizable signature (unforgeability), the probability of each adversary, who does not possess the attributes' set $\mathbb{S}$ such that $\mathbb{A}(\mathbb{S}) = 0$, is negligible.

Then, we define a game between the challenger $\mathscr{C}$ and the adversary $\mathscr{A}_2$.

Game 2: in Game 2, both the challenger $\mathscr{C}$ and the adversary $\mathscr{A}_2$ perform as defined in the security definition.

(i) Setup phase: the adversary $\mathscr{A}_2$ does as in the "Threat Model."

(ii) Query phase: the adversary $\mathscr{A}_2$ does as the adversary $\mathscr{A}_2$ in the query phase.

(iii) Challenge phase: the adversary $\mathscr{A}_2$ adaptively chooses the authorized user's attributes' set $\mathbb{S}$ ($\mathbb{A}(\mathbb{S}) = 1$). Then, $\mathscr{A}_2$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm to generate the challenged signature σ^* for the challenged message m^* which does not contain all inadmissible blocks. Finally, the adversary $\mathscr{A}_2$ sends $(\mathbb{S}, m^*, \sigma^*)$ to $\mathscr{C}$.

(iv) Verify phase: the adversary $\mathscr{A}_2$ performs polynomial queries as in the query phase. Consider the adversary $\mathscr{A}_2$ has made L queries, and let $Q = \{\text{sk}_{\mathbb{S}}, \mathbb{S}, m_i, A_i, \mathbb{A}_i, \sigma_i\}_{i=1}^{[|Q|]}$ denote the set of information obtained through these queries. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}_{\text{P3S}}^{\text{Sig}}, A, \mathbb{A}, m^*, \sigma^*)$ algorithm and outputs a bit b_0. If $b_0 = 1$, $\mathscr{C}$ checks whether there exists an $i \in [|Q|], m^*$ which does not contain all inadmissible blocks. If there is such an i, the challenger $\mathscr{C}$ outputs $b_1 = 1$. Otherwise, $\mathscr{C}$ outputs $b_1 = 0$.

Suppose $b_1 = 1$, that is, the adversary $\mathscr{A}_2$ wins, we can say that the adversary $\mathscr{A}_2$ breaks the security of the policy-based sanitizable signature because the adversary's goal is to correctly generate the valid signature σ' for the transaction m^*. According to the security of the policy-based sanitizable signature (immutability), the probability of each adversary, who redacts the inadmissible blocks, is negligible.

In conclusion, the proposed blockchain protocol achieves controlled redaction. In other words, only authorized users can redact the admissible portions of the transaction T_i. □

Theorem 3 (accountability). In the proposed blockchain protocol, trusted authority (group manager) can extract the identity of the originator of the transaction or the authorized user from any valid witness with nonnegligible probability.

Proof. We prove accountability by a sequence of games.

(i) Game 0: as Game 0 in [16].

(ii) Game 1: as Game 0, but we replace crs_Ω with the one generated by $(\text{crs}_\Omega, \tau) \leftarrow \text{SIM}_1(1^\kappa)$, i.e., the simulator SIM_1 takes the security parameter 1^κ as the input and then outputs $(\text{crs}_\Omega, \tau)$. Finally, the challenger $\mathscr{C}$ keeps the trapdoor τ and starts simulating all proofs.

(iii) Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathscr{B}$ which breaks the zero-knowledge property of the

underlying proof system. The reduction works as follows. Our adversary $\mathscr{B}$ receives crs_Ω from its own challenger and embeds it into $\mathrm{PP}_{\mathrm{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle provided and embedded honestly. Then, whatever $\mathscr{A}$ outputs is also output by $\mathscr{B}$. $|\Pr[S_0] - \Pr[S_1]|$ is negligible, where $\Pr[S_X]$ denotes the advantage of the adversary in Game X. Note that this also means that all proofs are now simulated, even though they still prove valid statements.

(iv) Game 2: as Game 1, but we replace crs_Ω with the one generated by $(\mathrm{crs}_\Omega, \tau, \xi) \leftarrow \xi_1(1^\kappa)$, i.e., the simulator ξ_1 takes the security parameter 1^κ as the input and then outputs $(\mathrm{crs}_\Omega, \tau, \xi)$. Finally, the challenger $\mathscr{C}$ keeps the trapdoors τ and ξ. Let E_2 be the event that $\mathscr{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that, by definition, crs_Ω is exactly distributed as in the prior hop.

(v) As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]|$ is negligible.

(vi) Game 3: as Game 2, but we abort if the adversary outputs valid $(\mathrm{pk}^*, m^*, \sigma^*)$ for which we cannot (as the holder of $\mathrm{sk}_{\mathrm{P3S}}^{\mathrm{Sig}}$) calculate pk which makes $\mathrm{Judge}_{\mathrm{P3S}}(\mathrm{pk}^*, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{Sig}}, \mathrm{pk}, \pi_{\mathrm{P3S}}, \sigma^*, m^*)$ output 0. Let this event be E_3.

If E_3 occurs, we have a bogus proof π contained in σ^* as it proves a false statement. Thus, $\mathscr{B}$ proceeds as in the prior game (doing everything honestly, but using simulated proofs and simulated crs_Ω) and can simply return the statement claimed to be proven by π and π itself. $|\Pr[S_2] - \Pr[S_3]|$ is negligible.

In conclusion, the proposed blockchain protocol achieves accountability. □

7. Performance

In this section, we first give functionality comparison among our redactable blockchain protocol and several related redactable blockchain protocols [11, 13–15]. Then, we analyze the computational burden of our redactable blockchain protocol through several experiments.

7.1. Functionality Comparison. We give functionality comparison among our scheme and the related schemes [11, 13–15]. As shown in Table 2, our scheme is the only one that satisfies all of the following properties: fine-grained access control, controllable edit, accountability, and supporting the redaction of both additional information and UTXO. The schemes in [11, 14] cannot support fine-grained access control. The scheme in [13] cannot effectively support harmful data deletion. All of these related redactable blockchain protocols cannot support controllable edit, accountability, and the editing of both additional information and UTXO.

7.2. Proof-of-Concept Implementation. To evaluate the practicality of the proposed blockchain protocol, we implement a full-fledged blockchain system in Python 3.5.3, which is carried out on a desktop with an Intel Core (TM) i5-4300 CPU @ 2.13 GHz and 8.0 GB RAM.

The blockchain system can achieve all the basic functionalities of Ethereum Enterprise. Separately, the blockchain system, including a subset of Ethereum Enterprise's script language, allows the authorized user to redact the transaction and the miner to delete the harmful data. We rely on the PoW consensus mechanism as Ethereum Enterprise does.

We evaluate the performance of the blockchain system for chain validation in different scenarios. In order to measure the cost time of chain validation, we validate chains containing different number of blocks and redaction transactions. A new chain is created and validated 50 times in each experiment, and the cost time of chain validation is the arithmetic mean of the run time of all runs. Each chain consists of up to 50,000 blocks, which approximate a one-year snapshot of the Ethereum Enterprise. Each block includes 1000 transactions (Figures 11–14).

(i) *Overhead Compared to the Immutable Blockchain.* In order to evaluate the overhead of the redactable blockchain protocol with no redactions performed compared to the immutable blockchain, in the series of experiments, the length of chains ranges from 10,000 to 50,000 blocks. As shown in Figure 11, the redactable blockchain protocol has only a more tiny overhead than the immutable blockchain. With the increase of the length of the chain, the overhead is smaller. The reason is that the only extra step of the redactable blockchain is to check if any votes are contained in the new block. The run time of this step is negligible compared to the time of chain validating when the length of the chain is larger enough.

(ii) *Overhead by the Number of Redactions.* In order to evaluate the overhead of the redactable blockchain protocol with the increasing number of redactions compared to the redactable blockchain with no redaction, in the series of experiments, the number of redactions ranges from 1000 to 5000. As shown in Figure 12, the overhead is linear in the number of redactions because we need to collect the votes for the redaction in the voting phase.

(iii) *Overhead by the Number of Removals.* In order to evaluate the overhead of the redactable blockchain protocol with the increasing number of removals compared to the redactable blockchain with no removal, in the series of experiments, the number of removals ranges from 1000 to 5000. As shown in Figure 13, the overhead is linear in the number of removals because the miner generating the new block needs to remove the harmful information from the previous block.

Table 2: Comparison of functionality among our redactable blockchain protocol and related protocols.

Protocols	Fine-grained access control	Controllable edit	Harmful data deletion	Accountability	Data type
[11]	×	×	√	×	Additional information
[13]	√	×	×	×	Additional information
[14]	×	×	√	×	Additional information
Ours	√	√	√	√	Additional information and UTXO

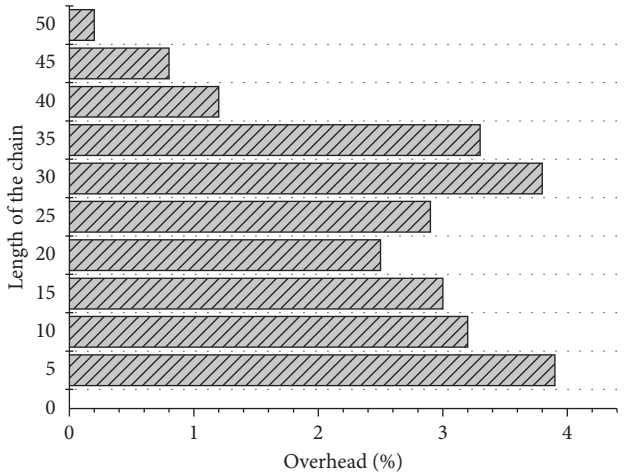

Figure 11: The overhead of the redactable blockchain without performing redaction compared to the immutable chain.

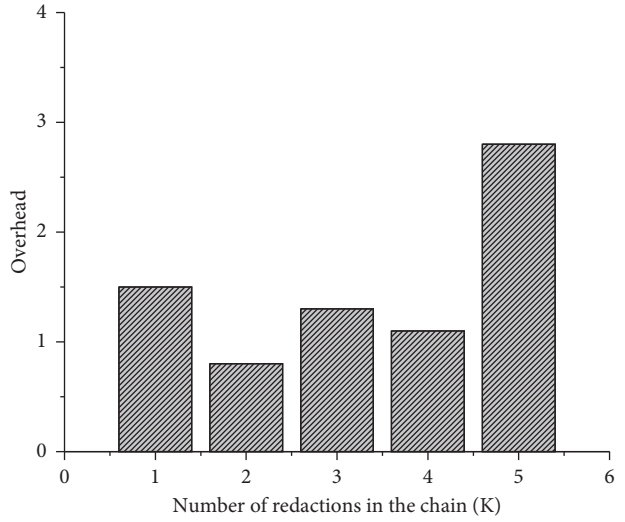

Figure 12: The overhead of the redactable blockchain for an increasing number of redactions compared to the redactable chain with no redaction.

(iv) *Overhead by the Voting Parameter ρ.* In order to evaluate the overhead of the redactable blockchain protocol with different voting periods, in the series of experiments, we set that the number of redactions is 1000, and the threshold ratio of the votes is $\rho \geq (1/2)$. As shown in Figure 14, the overhead is small and is most linear in the voting period.

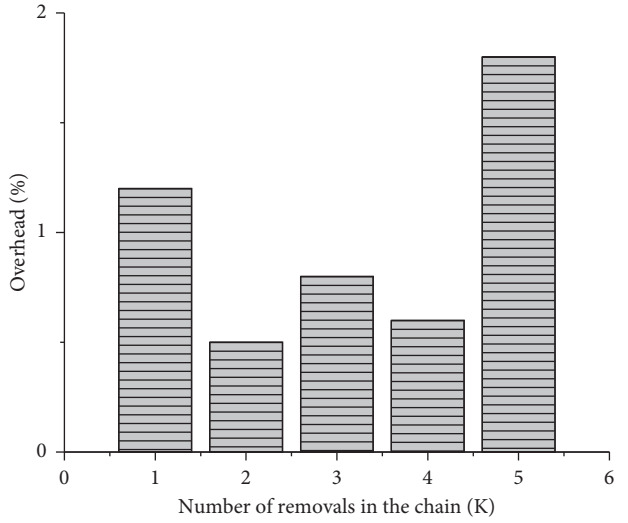

Figure 13: The overhead of the redactable blockchain for an increasing number of removals compared to the redactable chain with no removal.

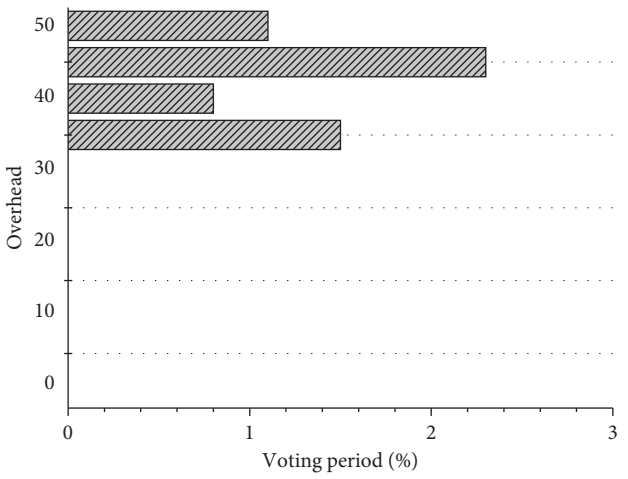

Figure 14: The overhead of the redactable blockchain for increasing voting periods compared to the redactable chain on a fixed voting period.

8. Conclusions

In this paper, we proposed a fine-grained and controllably redactable blockchain with harmful data forced removal. Our scheme not only supports the usual redaction of transactions but also the forced removal of harmful information in the blockchain. The originator of the transaction

could specify a fine-grained access control structure about who could redact the transaction and which portions of the transaction could be redacted. Any user could initiate a transaction that contains the index of the block which included harmful information without spending transaction fees. If the harmful information is contained in a block, it was forced to be deleted by the miner who created the new block. The user who provided the index of the block could receive the reward which was borne by the malicious user. The malicious user would be blacklisted if rewards were not paid within a period of time, and any transaction about the user would not be performed later. Furthermore, the scheme supported not only the redaction of additional data but also UTXO. Finally, we demonstrated that the scheme was secure and feasible via formal security analysis and proof-of-concept implementation.

Note that the proposed fine-grained and controllably redactable blockchain protocol with harmful data forced removal is suitable for permissioned blockchains, such as Hyperledger, Ethereum Enterprise, Ripple, and Quorum. There is another type of blockchain called permissionless blockchain, such as Bitcoin and Ethereum. Constructing the redactable permissionless blockchain protocol is a challenge and an interesting open problem. In our future work, we will also focus on designing more sophisticated solutions to the redactable permissionless blockchain protocol.

Appendix

A. Security Definition of the Improved Policy-Based Sanitizable Signature

In the following, we give the security definition of the improved policy-based sanitizable signature. Due to the limited space, we select several security aspects to highlight, and the rest of the security aspects can be seen in [16].

Definition 6. (unforgeability). In order to formally describe the unforgeability of the signature, we introduce a game between the challenger $\mathscr{C}$ and the adversary $\mathscr{A}$ to show how the adversary $\mathscr{A}$ is against the unforgeability of the signature. Trusted authority (group manager) is viewed as a challenger $\mathscr{C}$, and the unauthorized user is viewed as an adversary $\mathscr{A}$ in our security definition. This game includes the following phases:

(i) Setup phase: firstly, the challenger $\mathscr{C}$ runs the $\text{ParGen}_{\text{P3S}}$ and $\text{Setup}_{\text{P3S}}$ algorithm to generate the public parameters PP_{P3S} and the master private/public key pair $(\text{sk}_{\text{P3S}}, \text{pk}_{\text{P3S}})$. Then, $\mathscr{C}$ holds the master private key sk_{P3S} locally. Finally, $\mathscr{C}$ sends the master public key pk_{P3S} and the public parameters PP_{P3S} to the adversary $\mathscr{A}$.

(ii) Query phase:

(a) $\text{KGenSan}_{\text{P3S}}$ queries: the adversary $\mathscr{A}$ queries sanitizer's private/public key pair for the public parameters PP_{P3S}. $\mathscr{C}$ runs $\text{KGenSan}_{\text{P3S}}$ algorithm and returns the private/public key pair (x_2, y_2) to $\mathscr{A}$.

(b) Sign queries: the adversary $\mathscr{A}$ queries the signature for the master public key pk_{P3S}, the signature for the message m, the set of admissible blocks A, and the access structure $\mathbb{A}$. $\mathscr{C}$ runs $\text{KGenSig}_{\text{P3S}}$ to generate the signing key and then runs Sign algorithm to produce the signature σ. Finally, $\mathscr{C}$ returns the signature σ to $\mathscr{A}$.

(c) $\text{AddSan}_{\text{P3S}}$ queries: the adversary $\mathscr{A}$ queries the sanitizer's attribute key for sk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{San}}$, and the attributes' set $\mathbb{S}$ such that $\mathbb{A}(\mathbb{S}) = 0$. $\mathscr{C}$ runs $\text{AddSan}_{\text{P3S}}$ algorithm and returns the sanitizer's attribute key $\text{sk}_{\mathbb{S}} \leftarrow (\sigma_{\text{sk}_{\mathbb{S}}}, \text{sk}'_{\mathbb{S}})$ to $\mathscr{A}$.

(d) $\text{Verify}_{\text{P3S}}$ queries: the adversary $\mathscr{A}$ queries the verification result for pk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{A}$.

(e) $\text{Sanitize}_{\text{P3S}}$ queries: the adversary $\mathscr{A}$ queries the sanitizable signature for pk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{Sig}}$, $\text{sk}_{\text{P3S}}^{\text{San}}$, $\text{sk}_{\mathbb{S}}$, m, σ, and m'. $\mathscr{C}$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm and returns the new signature σ' to $\mathscr{A}$.

(f) $\text{Proof}_{\text{P3S}}$ queries: the adversary $\mathscr{A}$ queries $(\pi_{\text{P3S}}, \text{pk})$ for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Proof}_{\text{P3S}}$ algorithm and returns $(\pi_{\text{P3S}}, \text{pk})$ to $\mathscr{A}$.

(g) $\text{Judge}_{\text{P3S}}$ queries: the adversary $\mathscr{A}$ queries the judge result for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Judge}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{A}$.

(iii) Challenge phase: the adversary $\mathscr{A}$ adaptively chooses the authorized user's attributes' set $\mathbb{S}$ $(\mathbb{A}(\mathbb{S}) = 0)$. Then, $\mathscr{A}$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm to generate the challenged signature σ^* for the challenged message m^*. Finally, the adversary $\mathscr{A}$ sends $(\mathbb{S}, m^*, \sigma^*)$ to $\mathscr{C}$.

(iv) Verify phase: the adversary $\mathscr{A}$ performs polynomial queries as in the query phase. Consider the adversary $\mathscr{A}$ has made L queries, and let $Q = \{\text{sk}_{\mathbb{S}}, \mathbb{S}, m_i, A_i, \mathbb{A}_i, \sigma_i\}_{i=1}^{[|Q|]}$ denote the set of information obtained through these queries. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}_{\text{P3S}}^{\text{Sig}}, A, \mathbb{A}, m^*, \sigma^*)$ algorithm and outputs a bit b_0. If $b_0 = 1$, $\mathscr{C}$ checks whether there exists an $i \in [|Q|], \sigma^*$ such that $\mathbb{A}(\mathbb{S}) = 0$. If there is such an i, the challenger $\mathscr{C}$ outputs $b_1 = 1$. Otherwise, $\mathscr{C}$ outputs $b_1 = 0$.

We say that the adversary $\mathscr{A}$ wins if $b_1 = 1$. In the above game, we want to show that the adversary $\mathscr{A}$, who does not possess the attributes' set $\mathbb{S}$ such that $\mathbb{A}(\mathbb{S}) = 0$, should not generate the new valid signature. The adversary's goal is to correctly generate the valid signature σ' for the message m^*. We set the advantage of a polynomial-time adversary $\mathscr{A}$ in this game to be $\Pr[b_1 = 1]$. We say the proposed scheme satisfies the unforgeability of the signature if for any polynomial-time adversary $\mathscr{A}$, $\Pr[b_1 = 1] < (1/\text{poly}(n))$ for sufficiently large n, where poly stands for a polynomial function.

Definition 7. (immutability). In order to formally describe the immutability of the signed data, we introduce a game between the challenger $\mathscr{C}$ and the adversary $\mathscr{F}$ to show how the adversary $\mathscr{F}$ is against the immutability of the signed

data. Trusted authority (group manager) is viewed as a challenger $\mathscr{C}$, and the authorized sanitizer is viewed as an adversary $\mathscr{F}$ in our security definition. This game includes the following phases:

(i) Setup phase: firstly, the challenger $\mathscr{C}$ runs the $\text{ParGen}_{\text{P3S}}$ and $\text{Setup}_{\text{P3S}}$ algorithm to generate the public parameters PP_{P3S} and the master private/public key pair $(\text{sk}_{\text{P3S}}, \text{pk}_{\text{P3S}})$. Then, $\mathscr{C}$ holds the master private key sk_{P3S} locally. Finally, $\mathscr{C}$ sends the master public key pk_{P3S} and the public parameters PP_{P3S} to the adversary $\mathscr{F}$.

(ii) Query phase:

(a) $\text{KGenSan}_{\text{P3S}}$ queries: the adversary $\mathscr{F}$ queries sanitizer's private/public key pair for the public parameters PP_{P3S}. $\mathscr{C}$ runs $\text{KGenSan}_{\text{P3S}}$ algorithm and returns the private/public key pair (x_2, y_2) to $\mathscr{F}$.

(b) Sign queries: the adversary $\mathscr{F}$ queries the signature for the master public key pk_{P3S}, the signature for the message m, the set of admissible blocks F, and the access structure $\mathbb{A}$. $\mathscr{C}$ runs $\text{KGenSig}_{\text{P3S}}$ to generate the signing key and then runs Sign algorithm to produce the signature σ. Finally, $\mathscr{C}$ returns the signature σ to $\mathscr{F}$.

(c) $\text{AddSan}_{\text{P3S}}$ queries: the adversary $\mathscr{F}$ queries the sanitizer's attribute key for sk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{San}}$, and the attributes' set $\mathbb{S}$ such that $\mathbb{A}(\mathbb{S}) = 1$. $\mathscr{C}$ runs $\text{AddSan}_{\text{P3S}}$ algorithm and returns the sanitizer's attribute key $\text{sk}_{\mathbb{S}} \leftarrow (\sigma_{\text{sk}_{\mathbb{S}}}, \text{sk}_{\mathbb{S}}')$ to $\mathscr{F}$.

(d) $\text{Verify}_{\text{P3S}}$ queries: the adversary $\mathscr{F}$ queries the verification result for pk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{F}$.

(e) $\text{Sanitize}_{\text{P3S}}$ queries: the adversary $\mathscr{F}$ queries the sanitizable signature for pk_{P3S}, $\text{pk}_{\text{P3S}}^{\text{Sig}}$, $\text{sk}_{\text{P3S}}^{\text{San}}$, $\text{sk}_{\mathbb{S}}$, m, σ, and m'. $\mathscr{C}$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm and returns the new signature σ' to $\mathscr{F}$.

(f) $\text{Proof}_{\text{P3S}}$ queries: the adversary $\mathscr{F}$ queries $(\pi_{\text{P3S}}, \text{pk})$ for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Proof}_{\text{P3S}}$ algorithm and returns $(\pi_{\text{P3S}}, \text{pk})$ to $\mathscr{F}$.

(g) $\text{Judge}_{\text{P3S}}$ queries: the adversary $\mathscr{F}$ queries the judge result for pk_{P3S}, $\text{sk}_{\text{P3S}}^{\text{Sig}}$, σ, and m. $\mathscr{C}$ runs $\text{Judge}_{\text{P3S}}$ algorithm and returns the result to $\mathscr{F}$.

(iii) Challenge phase: the adversary $\mathscr{F}$ adaptively chooses the authorized user's attributes' set $\mathbb{S}$ $(\mathbb{A}(\mathbb{S}) = 1)$. Then, $\mathscr{F}$ runs $\text{Sanitize}_{\text{P3S}}$ algorithm to generate the challenged signature σ^* for the challenged message m^* which does not contain all inadmissible blocks. Finally, the adversary $\mathscr{F}$ sends $(\mathbb{S}, m^*, \sigma^*)$ to $\mathscr{C}$.

(iv) Verify phase: the adversary $\mathscr{F}$ performs polynomial queries as in the query phase. Consider the adversary $\mathscr{F}$ has made L queries, and let $Q = \{\text{sk}_{\mathbb{S}}, \mathbb{S}, m_i, A_i, \mathbb{A}_i, \sigma_i\}_{i=1}^{[|Q|]}$ denote the set of information obtained through these queries. $\mathscr{C}$ runs $\text{Verify}_{\text{P3S}}(\text{pk}_{\text{P3S}}, \text{pk}_{\text{P3S}}^{\text{Sig}}, A, \mathbb{A}, m^*, \sigma^*)$ algorithm and outputs a bit b_0. If $b_0 = 1$, $\mathscr{C}$ checks whether there exists an $i \in [|Q|], m^*$ which does not contain all inadmissible blocks. If there is such an i, the challenger $\mathscr{C}$ outputs $b_1 = 1$. Otherwise, $\mathscr{C}$ outputs $b_1 = 0$.

We say that the adversary $\mathscr{F}$ wins if $b_1 = 1$. In the above game, we want to show that the adversary $\mathscr{F}$, who redacts the inadmissible blocks, should not generate the new valid signature. The adversary's goal is to correctly generate the valid signature σ' for the message m^*. We set the advantage of a polynomial-time adversary $\mathscr{F}$ in this game to be $\Pr[b_1 = 1]$. We say the proposed scheme satisfies the unforgeability of the signature if for any polynomial-time adversary $\mathscr{F}$, $\Pr[b_1 = 1] < (1/\text{poly}(n))$ for sufficiently large n, where poly stands for a polynomial function.

Definition 8. (traceability). We say an improved policy-based sanitizable signature supports traceability if the trusted authority (group manager) can extract signer's identity from any valid signature with nonnegligible probability.

B. Security Analysis of the Improved Policy-Based Sanitizable Signature

In this section, we analyze the security of the improved policy-based sanitizable signature in terms of unforgeability, immutability, and traceability.

Theorem 4 (unforgeability). Any PPT adversaries can forge a policy-based sanitizable signature for some message with negligible probability.

Proof. To prove unforgeability, we use a sequence of games:

(i) Game 0: as Game 0 in [16].

(ii) Game 1: as Game 0, but we replace crs_{Ω} with the one generated by $(\text{crs}_{\Omega}, \tau) \leftarrow \text{SIM}_1(1^{\kappa})$, i.e., the simulator SIM_1 takes the security parameter 1^{κ} as the input and then outputs $(\text{crs}_{\Omega}, \tau)$. Finally, the challenger $\mathscr{C}$ keeps the trapdoor τ and starts simulating all proofs.

Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathscr{B}$ which breaks the zero-knowledge property of the underlying proof system. The reduction works as follows. Our adversary $\mathscr{B}$ receives crs_{Ω} from its own challenger and embeds it into PP_{P3S} and generates all other values honestly. All proofs are then generated using the oracle provided and embedded honestly. Then, whatever $\mathscr{A}$ outputs is also output by $\mathscr{B}$. $|\Pr[S_0] - \Pr[S_1]|$ is negligible. Note that this also means that all proofs are now simulated, even though they still prove valid statements.

(iii) Game 2: as Game 1, but we replace crs_{Ω} with the one generated by $(\text{crs}_{\Omega}, \tau, \xi) \leftarrow \xi_1(1^{\kappa})$, i.e., the simulator ξ_1 takes the security parameter 1^{κ} as the input and then outputs $(\text{crs}_{\Omega}, \tau, \xi)$. Finally, the challenger $\mathscr{C}$

keeps the trapdoors τ and ξ. Let E_2 be the event that $\mathcal{A}$ can distinguish this replacement with non-negligible probability. Moreover, note that, by definition, crs_Ω is exactly distributed as in the prior hop.

As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]|$ is negligible.

(iv) Game 3: as Game 2, but we abort if the adversary was able to generate a signature σ_m^* on a string never generated by the signing oracle. Let this event be E_3.

Assume, towards contradiction, that event E_3 occurs. We can then construct an adversary $\mathcal{B}$ which breaks the unforgeability of the underlying signature scheme, namely, $\mathcal{B}$ receives pk of the signature scheme. This is embedded in pk'_Σ, while all other values are generated as in Game 2. All oracles are simulated honestly, but $\mathrm{Sign}'_{\mathrm{P3S}}$. The only change is, however, that the generation of each σ_m is outsourced to the signature generation oracle. Then, whenever E_3 occurs, $\mathcal{B}$ can return $((\mathrm{pk}_{\mathrm{P3S}}, \mathrm{pk}^{\mathrm{Sig}}_{\mathrm{P3S}}, A, H(i\|m_{!A}), h, \mathbb{A}), \sigma_m^*)$. These values can easily be compiled using $\mathcal{A}$'s output, i.e., (m^*, σ^*). Note that this already includes that the adversary cannot temper with A. $|\Pr[S_2] - \Pr[S_3]|$ is negligible.

(v) Game 4: as Game 3, but we abort if the adversary was able to generate (m^*, σ^*) for which m^* should not have been derivable. Let this event be E_4.

Assume, towards contradiction, that event E_4 occurs. We can then construct an adversary $\mathcal{B}$ which breaks the strong insider collision resistance of the used PCH, namely, $\mathcal{B}$ receives $\mathrm{pk}_{\mathrm{PCH}}$ of the PCH. This is embedded in $\mathrm{pk}_{\mathrm{P3S}}$, while all other values are generated as in Game 3. The GetSan oracle is simulated honestly. Calls to the $\mathrm{Sign}'_{\mathrm{P3S}}$ oracle are done honestly, but the hash is generated using the $\mathrm{Hash}'_{\mathrm{PCH}}$ oracle. Calls to the $\mathrm{AddSan}'_{\mathrm{P3S}}$ oracle are simulated as follows. If a key for a simulated sanitizer (obtained by a call to the GetSan oracle) is to be generated, it is rerouted to $\mathrm{KGen}''_{\mathrm{PCH}}$. If the adversary wants to get a key for itself, it is rerouted to the $\mathrm{KGen}'_{\mathrm{PCH}}$ oracle, and the answer is embedded honestly in the response. Sanitization requests are performed honestly (but simulated proofs), with the exception that adaptions for simulated sanitizers are done using the $\mathrm{Adapt}'_{\mathrm{pch}}$ oracle. So far, the distributions are equal. Then, whenever the adversary outputs (m^*, σ^*) such that the winning conditions are fulfilled, our reduction $\mathcal{B}$ can return $(m^*, r^*, m'^*, r'^*, h^*)$. The values can be compiled from (m^*, σ^*) and the transcript from the signing oracle (note that we already excluded that the adversary can temper with the hash h). $|\Pr[S_3] - \Pr[S_4]|$ is negligible.

(vi) Game 5: as Game 4, but we abort if the adversary was able to generate (m^*, σ^*) but has never made a call $\mathrm{AddSan}'_{\mathrm{P3S}}$. Let this event be E_5.

Assume, towards contradiction, that event E_5 occurs. We can then construct an adversary $\mathcal{B}$ which breaks the unforgeability of used Σ or the one-wayness of the used one-way function f, namely, $\mathcal{B}$ receives pk_Σ of Σ and f, and $f(x) = y$ from its own challenger. This is embedded in $\mathrm{pk}_{\mathrm{P3S}}$ (and, of course, the public parameters), while all other values are generated as in Game 4. y is embedded in $\mathrm{pk}^{\mathrm{Sig}}_{\mathrm{P3S}}$. For signing, the proofs are already simulated, and thus, x is not required to be known. For each call to $\mathrm{AddSan}_{\mathrm{P3S}}$ for keys for which the adversary knows the corresponding secret keys, $\mathcal{B}$ calls its signature oracle to obtain such a key. For simulated sanitizers, those signatures do not need to be obtained as the proofs are already simulated. Then, whenever the adversary outputs (m^*, σ^*), $\mathcal{B}$ extracts values $(x_1, x_2, sk_\Pi, \sigma')$. If $f(x_1) = y$, $\mathcal{B}$ can return x_1 to break the one-wayness of f. In the other case, $\mathcal{B}$ can return $((f(x_2), \mathrm{pk}_{\mathrm{P3S}}), \sigma')$ as its own forgery attempt for Σ. If extraction fails or a wrong statement was proven, SSE does not hold. A reduction is straightforward. $|\Pr[S_4] - \Pr[S_5]|$ is negligible. Now, the adversary can no longer win the unforgeability game; this game is computationally indistinguishable from the original game, which concludes the proof. □

Theorem 5 (immutability). For each PPT adversary, the advantage of generating valid signatures for altered immutable parts is negligible.

Proof. To prove immutability, we use a sequence of games:

(i) Game 0: as Game 0 in [16].

(ii) Game 1: as Game 0, and we abort if the adversary outputs $(\mathrm{pk}^*, m^*, \sigma^*)$ such that the winning conditions are met. Let this event be E_1.

Assume, towards contradiction, that event E_1 occurs. We can then build an adversary $\mathcal{B}$ which breaks the unforgeability of the used signature scheme, namely, we know that A (which also contains the length of the message and all nonmodifiable blocks along with their location), along with $\mathrm{pk}_{\mathrm{PCH}}$, is signed. As, however, by definition, the message m^* must be different from any derivable message, A w.r.t. $\mathrm{pk}_{\mathrm{PCH}}$ was never signed in this regard. Thus, $(\mathrm{pk}^*, \mathrm{pk}^{\mathrm{Sig}}_{\mathrm{P3S}}, A^*, H^*(i\|m_{!A}), h^*, \mathbb{A}^*)$ was never signed by the signer.

Constructing a reduction $\mathcal{B}$ is now straightforward. Our reduction $\mathcal{B}$ receives the public key pk'_Σ (along with the public parameters) from its own challenger. This public key is embedded as pk'_Σ. All other values are generated honestly. If a signature σ_m is to be generated, $\mathcal{B}$ asks its own oracle to generate that signature, embedding it into the response $\mathcal{A}$ receives. At some point, $\mathcal{A}$ returns $(\mathrm{pk}^*, m^*, \sigma^*)$. The forgery can be extracted as described above. $|\Pr[S_0] - \Pr[S_1]|$ is negligible. We stress that, by construction, a sanitizer always exists. Now, the adversary can no longer win the immutability game; this game is computationally indistinguishable from the original game, which concludes the proof. □

Theorem 6 (traceability). Trusted authority (group manager) can extract the identity of the originator of the

transaction or the authorized user from any valid witness with nonnegligible probability.

Proof. We prove traceability by a sequence of games:

(i) Game 0: as Game 0 in [16].

(ii) Game 1: as Game 0, but we replace crs_Ω with the one generated by $(\mathrm{crs}_\Omega, \tau) \leftarrow \mathrm{SIM}_1(1^\kappa)$, i.e., the simulator SIM_1 takes the security parameter 1^κ as the input and then outputs $(\mathrm{crs}_\Omega, \tau)$. Finally, the challenger $\mathscr{C}$ keeps the trapdoor τ and starts simulating all proofs.

Assume towards contradiction that the adversary behaves differently. We can then build an adversary $\mathscr{B}$ which breaks the zero-knowledge property of the underlying proof system. The reduction works as follows. Our adversary $\mathscr{B}$ receives crs_Ω from its own challenger and embeds it into $\mathrm{PP}_{\mathrm{P3S}}$ and generates all other values honestly. All proofs are then generated using the oracle provided and embedded honestly. Then, whatever $\mathscr{A}$ outputs is also output by $\mathscr{B}$. $|\Pr[S_0] - \Pr[S_1]|$ is negligible. Note that this also means that all proofs are now simulated, even though they still prove valid statements.

(iii) Game 2: as Game 1, but we replace crs_Ω with the one generated by $(\mathrm{crs}_\Omega, \tau, \xi) \leftarrow \xi_1(1^\kappa)$, i.e., the simulator ξ_1 takes the security parameter 1^κ as the input and then outputs $(\mathrm{crs}_\Omega, \tau, \xi)$. Finally, the challenger $\mathscr{C}$ keeps the trapdoors τ and ξ. Let E_2 be the event that $\mathscr{A}$ can distinguish this replacement with nonnegligible probability. Moreover, note that, by definition, crs_Ω is exactly distributed as in the prior hop.

As we only keep one additional value, i.e., ξ, this is only an internal change. $|\Pr[S_1] - \Pr[S_2]|$ is negligible.

(iv) Game 3: as Game 2, but we abort if the adversary outputs valid $(\mathrm{pk}^*, m^*, \sigma^*)$ for which we cannot (as the holder of $\mathrm{sk}_{\mathrm{P3S}}^{\mathrm{Sig}}$) calculate pk which makes $\mathrm{Judge}_{\mathrm{P3S}}(\mathrm{pk}^*, \mathrm{pk}_{\mathrm{P3S}}^{\mathrm{Sig}}, \mathrm{pk}, \pi_{P3S}, \sigma^*, m^*)$ output 0. Let this event be E_3.

If E_3 occurs, we have a bogus proof π contained in σ^* as it proves a false statement. Thus, $\mathscr{B}$ proceeds as in the prior game (doing everything honestly, but using simulated proofs and simulated crs_Ω) and can simply return the statement claimed to be proven by π and π itself. $|\Pr[S_2] - \Pr[S_3]|$ is negligible. □

Data Availability

We thank the authors of [14] for providing their implementation to us. We emailed Dominic Deuber and Bernardo Magri and obtained the source code for their scheme named "Redactable Blockchain in the Permissionless Setting." [14] We then extended and improved the source code to implement our scheme. We cannot expose the source code of the scheme in [14] without the permission of its authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

We thank the authors of [14] for providing their implementation to us. This work was supported by the National Natural Science Foundation of China (Grant nos. U1536205, 61472084, 61972094, and 62032005), National Key Research and Development Program of China (Grant no. 2017YFB0802000), Shanghai Innovation Action Project (Grant no. 16DZ1100200), Shanghai Science and Technology Development Funds (Grant no. 16JC1400801), Shandong Provincial Key Research and Development Program of China (Grant nos. 2017CXGC0701 and 2018CXGC0701), and the Young Talent Promotion Project of Fujian Science and Technology Association.

References

[1] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008, https://nakamotoinstitute.org/bitcoin/.

[2] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: a technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[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 ACM SIGSAC on Computer and Communications Security*, pp. 913–930, Toronto Canada, October 2018.

[4] L. Breidenbach, I. Cornell Tech, P. Daian, F. Tramer, and A. Juels, "Enter the hydra: towards principled bug bounties and exploit-resistant smart contracts," in *Proceedings of the 27th USENIX Security*, Baltimore, MD, USA, August 2018.

[5] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: analysis and applications," in *Proceedings of the EUROCRYPT 2015*, pp. 281–310, Sofia, Bulgaria, April 2015.

[6] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: a provably secure proof-of-stake blockchain protocol," in *Proceedings of the Annual International Cryptology Conference*, pp. 357–388, Santa Barbara, CA, USA, August 2017.

[7] L. D. Ibanez, K. O'Hara, and E. Simperl, "On blockchains and the general data protection regulation," *European Parliament Think Tank*, 2018, https://www.europarl.europa.eu/thinktank/en/document.html?reference=EPRS_STU%282019%29634445.

[8] Interpol, "Interpol cyber research identifies malware threat to virtual currencies," 2015, https://www.interpol.int/News-and-Events/News/2015/INTERPOL-cyber-research-identifies-malware-threat-to-virtual-currencies.

[9] G. Tziakouris, "Cryptocurrenciesła forensic challenge or opportunity for law enforcement? an interpol perspective," in *Proceedings of the IEEE Security & Privacy*, vol. 16, no. 4, pp. 92–94, San Francisco, CA, USA, May 2018.

[10] R. Matzutt, J. Hiller, M. Henze et al., "A quantitative analysis of the impact of arbitrary blockchain content on bitcoin," in *Proceedings of the 22nd FC*, Nieuwpoort, Curaçao, February 2018.

[11] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchainCorCrewriting history in bitcoin and

friends," in *Proceedings of the Euro S & P 2017*, pp. 111–126, Paris, France, April 2017.

[12] J. Camenisch, D. Derler, S. Krenn, and H. C. P, hls, K. Samelin, and D. Slamanig, "Chameleon-hashes with ephemeral trapdoors," in *Proceedings of the IACR PKC*, Amsterdam, The Netherlands, March 2017.

[13] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-grained and controlled rewriting in blockchains: chameleon-hashing gone attribute-based," in *Proceedings of NDSS*, San Diego, CA, USA, February 2019.

[14] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable blockchain in the permissionless setting," in *Proceedings of the SP2019*, pp. 19–23, San Francisco, CA, USA, May 2019.

[15] M. Florian, S. Henningsen, S. Beaucamp, and B. Scheuermann, "Erasing data from blockchain nodes," in *Proceedings of the EuroS&P*, pp. 367–376, Stockholm, Sweden, June 2019.

[16] K. Samelin and D. Slamanig, "Policy-based sanitizable signatures," in *Proceedings of the CT-RSA 2020*, pp. 538–563, San Francisco, CA, USA, February 2020.

[17] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik, "Sanitizable signatures," in *Proceedings of the ESORICS 2005*, vol. 3679, pp. 159–177, Milan, Italy, September 2005.

[18] C. Brzuska, M. Fischlin, T. Freudenreich et al., "Security of sanitizable signatures revisited," in *Proceedings of the PKC 2009*, pp. 317–336, Irvine, CA, USA, March 2009.

[19] C. Brzuska, M. Fischlin, A. Lehmann, and Schr, D. der, "Unlinkability of sanitizable signatures," in *Proceedings of the PKC 2010*, pp. 444–461, Paris, France, May 2010.

[20] S. Canard, F. Laguillaumie, and M. Milhau, "Trapdoor sanitizable signatures and their application to content protection," in *Proceedings of the ACNS 2008*, pp. 258–276, New York, NY, USA, June 2008.

[21] J. Lai, X. Ding, and Y. Wu, "Accountable trapdoor sanitizable signatures," in *Proceedings of the ISPEC 2013*, pp. 117–131, Lanzhou, China, May 2013.

[22] K. Miyazaki, G. Hanaoka, and H. Imai, "Digitally signed document sanitizing scheme based on bilinear maps," in *Proceedings of the 2006 ACM Conference on Computer and Communications Security*, pp. 343–354, Alexandria, VA, USA, October 2006.

[23] T. H. Yuen, W. Susilo, J. K. Liu, and Y. Mu, "Sanitizable signatures revisited," in *Proceedings of the CANS 2008*, pp. 80–97, Hong-Kong, China, December 2008.

[24] S. Agrawal, S. Kumar, A. Shareef, and C. P. Rangan, "Sanitizable signatures with strong transparency in the standard model," in *Proceedings of the Inscrypt 2009*, pp. 93–107, Shanghai, China, October 2010.

[25] J. Ning, X. Huang, W. Susilo, K. Liang, X. Liu, and Y. Zhang, "Dual access control for cloud-based data storage and sharing," *IEEE Transactions on Dependable and Secure Computing*, vol. 99, p. 1, 2020.

[26] J. Ning, Z. Cao, X. Dong, K. Liang, L. Wei, and K. R. Choo, "CryptCloud+: secure and expressive data access control for cloud storage," *IEEE Transactions on Service Computing*, vol. 99, p. 1, 2018.

[27] J. Ning, Z. Cao, X. Dong, H. Ma, L. Wei, and K. Liang, "Auditable σ-times outsourced attribute-based encryption for access control in cloud computing," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 94–105, 2018.

[28] X. Liu, J. Ma, J. Xiong, J. Ma, and Q. Li, "Attribute based sanitizable signature scheme," *Journal of Communications*, vol. 34, pp. 148–155, 2013.

[29] L. Xu, X. Zhang, X. Wu, and W. Shi, "ABSS: an attribute-based sanitizable signature for integrity of outsourced database with public cloud," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 167–169, San Antonio, TX, USA, March 2015.

[30] R. Mo, J. Ma, X. Liu, and Q. Li, "FABSS: attribute-based sanitizable signature for flexible access structure," in *Proceedings of the ICICS 2017*, Beijing, China, December 2017.