Decentralized and Efficient Blockchain Rewriting with Bi-level Validity Verification

Kemin Zhang[1], Li Yang[1(✉)], Lu Zhou[1], and Jianfeng Ma[2]

[1] School of Computer Science and Technology, Xidian University, Xi'an, China
yangli@xidian.edu.cn
[2] School of Cyber Engineering, Xidian University, Xi'an, China

Abstract. Numerous studies have established that the immutability, a crucial property of blockchains, need to be delicately broken under certain circumstance as the content in blockchains could be compelled to redact for personal or legal reasons. Existing schemes ordinarily leverage policy-based chameleon hash (PCH) to perform fine-grained rewriting on blockchains, where modifiers with attributes satisfying the access policy can be authorized to modify the content in the blockchain. However, these schemes rely on a single trusted authority for managing rewriting permissions, which could be affected by a potential single point of failure. Meanwhile, heavy computations in such schemes might affect the performance in practical use.

To address these limitations, we propose a decentralized and efficient blockchain rewriting scheme with bi-level validity verification. With the integration of the multi-authorities attribute-based encryption, our scheme supports the modifier to obtain rewriting secret keys from various authorities for performing rewriting at transaction level. Moreover, computationally intensive operations in our scheme can be performed in stages and partially outsourced to the proxy server. As an assurance of security, our scheme provides bi-level validity verification for the rewriting secret key and the content on blockchain. Moreover, we present formal security analysis and conduct comparison experiments to illustrate the advantages in both functionality and performance.

Keywords: Blockchain rewriting · Attribute-based encryption · Chameleon hash · Outsource computing

1 Introduction

The concept of the blockchain was originally introduced by Satoshi Nakamoto to function as a distributed public ledger of the cryptocurrency Bitcoin [23]. It has raised a widespread concern in the community and has found wide applications in many fields such as supply chain [11], Internet-of-Things (IoT) [25] and healthcare services [8]. A blockchain is a continuously expanding list of records made

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 309–328, 2023.
https://doi.org/10.1007/978-3-031-25538-0_17

up of cryptographically connected blocks, which is resistant to modifications since the transaction data in any fixed block cannot be modified retroactively without editing all subsequent blocks.

However, as a crucial property of the blockchain, the immutability could be a limiting factor for practical promoting in certain circumstance. From the perspective of privacy protection, the immutability is inherently incompatible with certain regulations that emphasize protecting user privacy and avoiding sensitive content, for instance, General Data Protection Regulation (GDPR) [30], since any data (e.g. transaction values) in such immutable blockchains cannot be erased. Moreover, the illicit content in blockchains such as pornography, violent narratives or viruses uploaded by malicious users could cause lasting and negative impact. Therefore, an urgent demand for schemes of redacting incorrect or even illicit contents in the blockchain is desirable in practical scenarios.

Earlier studies intended to perform block-level rewriting on blockchains, predominately by replacing the traditional hash function with a trapdoor-based chameleon hash [16]. Subsequently, for providing transaction-level redaction, several studies aimed to implement fine-grained and controlled blockchain rewriting by employing attribute-based encryption (ABE) [9,14,28,33]. In these schemes, a predetermined access policy is embedded into a transaction and the modifier possessing a trapdoor could find hash collisions to modify the transaction if her attributes satisfy the embedded access policy.

Nevertheless, the aforementioned fine-grained blockchain rewriting schemes are still flawed to some extent. Specifically, these schemes either rely on a trusted authority for key distribution, which could be affected by a potential single point of failure, or necessitate heavy computation which might affect the performance in practical use. The requirement for a single trusted authority could be incompatible with the decentralized designing of the blockchain. Users with resource-constrained devices could not efficiently perform complex computations such as bilinear pairing operations in the hash or decryption phase. Moreover, the validity of the rewriting secret key received by the modifier and the transaction content cannot be verified.

To address these limitations, we propose a scheme of decentralized and efficient blockchain rewriting with bi-level validity verification. In our scheme, the transaction owner appends signatured transactions with the embedded access policy to the blockchain. Subsequently, the modifier obtains rewriting secret keys from multiple authorities and concurrently performs validity verification. As a consequence, the distribution of the rewriting secret key is decentralized which accordingly eliminates the risk of a potential single point of failure. Finally, the modifier requests partial decryption from the proxy server and further completes the decryption to rewrite and sign the transaction in the blockchain. Note that the computation of both hash and decryption are separated into two stages in our scheme. For this reason, the computational burden on the user side is significantly reduced, which is beneficial for practical implementation.

To the best of our knowledge, this scheme is the first to simultaneously address the aforementioned limitations, and the contributions of the work are summarized in the following four aspects:

- We introduce a framework of decentralized blockchain rewriting scheme that allows for transaction-level rewriting, based on multi-authority ABE. The modifier can perform transactions rewriting only if her attributes satisfy the access policy predetermined by the transaction owner, where the privilege to the modifier for rewriting is granted jointly by multiple authorities.
- We adopt offline/online hashing and outsourced computing in our scheme for reducing the computational burden on the user side. Specifically, the hash algorithm is separated into offline/online phases, and the decryption to be executed by the modifier is split into two phases for performing outsourced partial decryption.
- We provide a mechanism of bi-level validity verification, in order for the modifier to verify the validity of rewriting secret keys received from various attribute authorities, as well for any entity to verify the content in the blockchain.
- We build an instantiation of our scheme on firm theoretical grounds. All important properties of our scheme are formalized in Sect. 3, and the security of the proposed scheme is demonstrated via formal security analysis in Sect. 4.3. Besides, the experimental results of the comparison with previous schemes demonstrate that our scheme is advantageous in both functionality and performance.

1.1 Related Work

Attribute-based Encryption. Sahai and Waters [26] first introduced the concept of a public-key encryption scheme, namely attribute-based encryption (ABE), in which ciphertexts and secret keys are dependent upon attributes. Subsequently, Goyal et al. [12] developed a cryptosystem named *key-policy attribute-based encryption* (KP-ABE), in order to perform fine-grained sharing of encrypted data. In this scheme, ciphertexts are labeled with attributes and can be decrypted by private keys with respect to access structures. Bethencourt et al. [3] presented a system named *ciphertext-policy attribute-based encryption* (CP-ABE) to perform access control on the encrypted data. Guo et al. [13] introduced *identity-based offline/online encryption* (IBOOE), which separates identity-based encryption into online and offline phases for improving the computing efficiency. Afterwards, extensive research on KP-ABE or CP-ABE were proposed for better efficiency or security [7,15,17–19,21,31,32]. For instance, in [15], the vast majority of the work on encryption or secret key generation in ABE is performed offline to reduce the cost in practice. In [18], Lewko and Waters proposed a multi-authority ABE system which allows any party to become an authority, and thus avoids the performance bottleneck incurred by relying on a central authority.

Blockchain Rewriting. Blockchain rewriting has raised widespread concern in the community since the pioneering work [2]. In this work, chameleon hash (CH) function are deployed instead of the original SHA256 hash function for rewriting block contents in blockchains. Chameleon hash is collision resistant if the trapdoor is unknown, conversely, a modifier who is aware of the trapdoor can find collisions and perform rewriting operations while keeping the hash value.

Subsequently, Puddu et al. [24] proposed a mutable blockchain, named μchain, which provides mechanisms for removing record data from the blockchain and their modifications. Thyagarajan et al. [27] introduced a publicly verifiable layer *Reparo* to fix incorrect contracts and remove illicit contents from the blockchain. Deuber et al. [10] proposed an efficient redactable blockchain for permissionless setting based on consensus-based voting. It dispenses with sophisticated cryptographic techniques or trust assumptions. However, the block-level rewriting operation is coarse-grained, i.e. the whole block have to be replaced even only one transaction in a block is required to be modified.

To bridge this gap, a line of studies have been proposed to achieve the goal of fine-grained and controlled rewriting based on ABE scheme. For instance, Derler et al. [9] presented *policy-based chameleon hashes* (PCH) that integrate CP-ABE with *chameleon hash with ephemeral trapdoor* (CHET) [5] to perform fine-grained modifications on blockchains. Any modifier own the attributes which satisfy the access policy can find hash collisions and further rewrite the blockchain at transaction-level. In [28], the scheme of *policy-based chameleon hash with black-box accountability* (PCHBA) was proposed to identify responsible transaction modifiers in case of dispute while achieving the goal of fine-grained rewriting on blockchain. Subsequently, the authors further generalized their work to a permissionless setting [29], which leverages *dynamic proactive secret sharing* (DPSS) [22] to remove the trusted authority and utilize KP-ABE for fine-grained access control. In [33], a *multi-authority policy-based chameleon hash* (MAPCH) was proposed by combing CHET and multi-authority CP-ABE, for reducing the workload of a single authority. In [14], a new rewritable blockchain scheme, named OO-RB-AOC, was proposed to reduce computational overhead and improve the security by performing an auditable outsourced computation mechanism for some time-consuming operations.

2 Preliminaries

2.1 Bilinear Mapping

$\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of prime order p. Then a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ meets the following properties:

- Bilinearity: $\forall x, y \in \mathbb{Z}_p$, $\forall \alpha, \beta \in \mathbb{G}_1$, then $e(\alpha^x, \beta^y) = e(\alpha, \beta)^{xy}$.
- Non-degeneracy: $\exists \alpha \in \mathbb{G}_1$, such that $e(\alpha, \alpha) \neq 1_{\mathbb{G}_2}$, where $1_{\mathbb{G}_2}$ is the identity element of $\mathbb{G}_2$.
- Computability: $\forall \alpha, \beta \in \mathbb{G}_1$, the value of $e(\alpha, \beta)$ can be computed efficiently.

2.2 Multi-authority Ciphertext-Policy Attribute-Based Encryption (CP-ABE)

Multi-authority CP-ABE allows any entity to become an authority, and multiple authorities are responsible for managing the attribute sets of users. In this paper, we adopt the multi-authority CP-ABE scheme proposed in [20]. It executes hash function on users' global identifier *gid* [7] to integrate private keys of

each user received form various authorities. This scheme simultaneously achieves the goals of autonomous key generation and collusion resistance. Specifically, multi-authority CP-ABE algorithm consists of the following five algorithms:

- *Global Setup*$(\lambda) \rightarrow GP$: Taking the security parameter λ as the input, it outputs global parameter GP for the system.
- *Authority Setup*$(GP) \rightarrow pk_{\mathrm{AA}}, sk_{\mathrm{AA}}$: Each authority runs the algorithm with GP to generate its own public and secret key pair (pk_{AA}, sk_{AA}).
- *KeyGen*$(gid, GP, i, sk_{\mathrm{AA}}) \rightarrow SK_{i,gid}$: Taking an authority identity gid, global parameter GP, an attribute i, and the secret key sk_{AA} as the input, the algorithm outputs $SK_{i,gid}$ for the attribute-identity pair.
- *Encrypt*$(GP, \{pk_{\mathrm{AA}}\}, \mathbb{A}, m) \rightarrow CT$: The algorithm takes the global parameter GP, the set of public keys for related authorities $\{pk_{\mathrm{AA}}\}$, access policy matrix $\mathbb{A} = (A, \rho)$, and a message m as the input, and outputs a ciphertext CT.
- *Decrypt*$(CT, GP, \{SK_{i,gid}\}) \rightarrow m$: It takes the ciphertext CT, the global parameter GP, and a set of keys $\{SK_{i,gid}\}$ as the input. If the collection of attribute i satisfies the access policy embedded in the ciphertext, the algorithm outputs the message m; otherwise the decryption cannot succeed.

2.3 Chameleon Hash (CH)

An arbitrary collision can be found in the domain of the function by using a trapdoor presented in chameleon hash functions [16]. On the message space $\mathcal{M}$, the chameleon hash employs the following four algorithms:

- *KeyGen*: The chameleon key pair (pk, sk) is computed by the security parameter λ in this algorithm.
- *Hash*: This algorithm calculates a chameleon hash h and a randomness r using chameleon secret key pk and a message m.
- *Verify*: The verify algorithm takes the chameleon public key pk, chameleon hash h, randomness r and message m as the input, and then outputs a bit $b \in \{0, 1\}$.
- *Adapt*: The chameleon private key sk, chameleon hash h, randomness r, original and new message (m, m') are taken as the input, then a new randomness value r' which could be used to recover the trapdoor is computed.

2.4 Bnoeh-Lynn-Shacham (BLS) Signature

We leverage BLS signature [4] to aggregate multiple signatures into a single signature without the interactions between each authority, in order for the modifier to verify the validity of the rewriting secret key. Specifically, the BLS signature algorithm consists of three algorithms listed below:

- *Ken*: This algorithm computes the public key c and the secret key a.
- *Sign*: It takes a and the message m as the input, and outputs the signature σ.
- *Ver*: This algorithm takes m, c and σ as the input, and outputs a value $d \in \{0, 1\}$ which indicates the correctness of signatures.

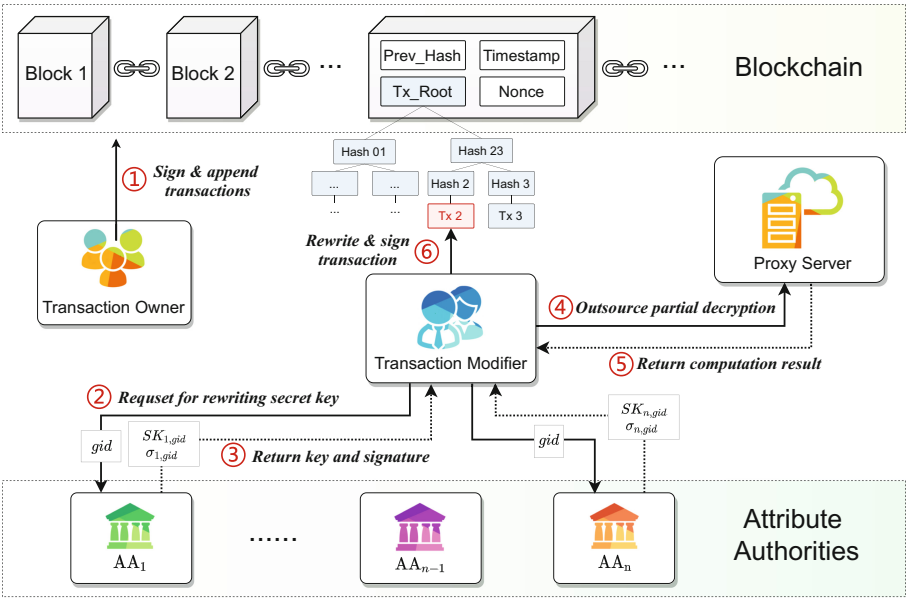

Fig. 1. System model

3 Models and Definitions

3.1 System Model

Our proposed system consists of four entities (shown in Fig. 1), namely Transaction Owner (TO), Attritube Authority (AA), Transaction Modifier (TM) and Proxy Server (PS). The interactions and functions of each entity are listed below:

Transaction Owner (TO): TO appends hashed transactions with signature to the blockchain. In our scheme, TO's devices are considered to be trusted and reliable yet could be resource-constrained (for instance, mobile phone).

Attribute Authority (AA): AAs are responsible for managing the attributes of users as well as generating and distributing the rewriting secret key associated with the TM's attribute set. There is no requirement for any global coordination amongst the authorities. Moreover, AA further signs the rewriting secret key in order for the TM to verify the validity.

Transaction Modifier (TM): The rewriting operation in blockchain is performed by the TM. Concretely, TM provides global identifier *gid* to collect rewriting secret keys corresponding to her attribute set from AAs, and subsequently verifies the validity of keys by performing signature aggregation. Afterwards, TM requests partial decryption from PS and completes the rest of the computation. TM could find hash collision to perform transaction rewriting once its attributes satisfy the access policy predetermined by TO.

Proxy Server (PS): PS performs partial decryption using transformation keys received from TM, and subsequently sends the result to the TM. As a consequence, the decryption load on TM's devices is reduced significantly.

3.2 Definition

Our proposed scheme is based on multi-authority ciphertext-policy attribute-based encryption (CP-ABE) and chameleon hash (CH) for decentralized rewriting on blockchain. It is comprised of the following five phases:

Setup: Arbitrary AA runs the *Global_Setup* algorithm to generate the global parameter. The *Auth_Setup* and *Modi_Setup* algorithms are used to generate the key pairs of all AAs and TM respectively;

KeyGen: Each AA runs *Rew_KeyGen* algorithm to generate and sign the rewriting secret keys associated with the attribute set of TM. Subsequently, TM runs *Trans_KeyGen* algorithm to generate and send the transformation keys to PS for outsourcing decryption;

Hash: TO firstly runs *Offline_Hash* algorithm to generate the intermediate ciphertext, which can be viewed as pre-computations for computing the final ciphertext. Afterwards, TO runs *Online_Hash* algorithm to generate chameleon hash, and subsequently signs the transaction for identification;

Verify: TM and any entity can respectively run *Aggre_Verify* and *Hash_Verify* algorithm, in order to verify the validity of the rewriting secret key and the transaction content;

Adapt: PS runs *Part_Decrypt* algorithm to execute partial decryption using transformation keys. TM subsequently runs *Full_Adapt* algorithm to find hash collision for rewriting, and signs the modified transaction for identification.

More precisely, the proposed scheme consists of the following algorithms:

- $Global\_Setup(\lambda) \rightarrow GP$. Any AA could run the algorithm to compute global parameter GP using security parameter λ.
- $Auth\_Setup(GP) \rightarrow (pk_{\mathrm{AA}}, sk_{\mathrm{AA}}, pk_{\mathrm{sig}}, sk_{\mathrm{sig}})$. In accordance with GP, each AA computes public key pk_{AA}, the secret key sk_{AA} and the key pair $(pk_{\mathrm{sig}}, sk_{\mathrm{sig}})$ for the signature.
- $Modi\_Setup(GP) \rightarrow (spk, ssk)$. TM runs this algorithm to obtain a key pair (spk, ssk) for signature using GP.
- $Rew\_KeyGen(GP, gid, i, sk_{\mathrm{AA}}, sk_{\mathrm{sig}}) \rightarrow (SK_{i,gid}, \sigma_{i,gid})$. An authority owning the attribute i inputs global parameter GP, global identity gid, attribute i and secret keys $(sk_{\mathrm{AA}}, sk_{\mathrm{sig}})$, and computes rewriting secret key $SK_{i,gid}$ and signature $\sigma_{i,gid}$ for an attribute-identity pair (i, gid).
- $Trans\_KeyGen(GP, SK_{i,gid}) \rightarrow TK_{i,gid}$. Given the global parameter GP, TM transforms the $SK_{i,gid}$ into a transformation key $TK_{i,gid}$ which will be sent to a PS for partial decryption.

- *Offline_Hash*$(GP) \rightarrow (IC, IS)$. TO computes intermediate ciphertext IC and intermediate state IS using GP. The offline hash phase could accelerate the construction of the final ciphertext since it can be seen as the pre-computation of the online hash phase (detailed in Sect. 4.1). For reducing computational consumption, TO could execute the algorithm in the charging or idle time of the device.
- *Online_Hash*$(GP, \{pk_{\mathrm{AA}}\}, IC, IS, \mathbb{A}, m) \rightarrow (h, r, \sigma_{\mathrm{user}})$. Given GP, a set of public keys $\{pk_{\mathrm{AA}}\}$ from multiple authorities, intermediate ciphertext IC, intermediate state IS, access policy $\mathbb{A} = (A, \rho)$ and message m, TO executes the algorithm to compute a hash h, a randomness r and a signature σ_{user}.
- *Aggre_Verify*$(gid, \{\sigma_{i,gid}\}, pk_{\mathrm{sig}}) \rightarrow 0$ or 1. TM could verify the validity of rewriting secret keys received from multiple authorities. Given gid of TM, signatures set $\{\sigma_{i,gid}\}$ from all authorities owning TM's attribute set, and authorities' public key pk_{sig} for signature verification, the $SK_{i,gid}$ is checked out to be valid if the algorithm returns 1, or else it returns 0.
- *Hash_Verify*$(m, h, r, \sigma_{\mathrm{user}}) \rightarrow 0$ or 1. Any entity could take m, (h, r) and signature σ_{user} to verify whether the hash h and signature σ_{user} is valid. The output is a bit $b \in \{0, 1\}$.
- *Part_Decrypt*$(GP, CT, \{TK_{i,gid}\}) \rightarrow CT'$. PS could use GP, ciphertext CT and transformation key $\{TK_{i,gid}\}$ to compute partial decryption result CT'.
- *Full_Adapt*$(h, r, m, m', \{TK_{i,gid}\}, \sigma_{\mathrm{user}}) \rightarrow (r', \sigma'_{\mathrm{user}})$. TM utilizes (h, r), m, new message m', $\{TK_{i,gid}\}$ and σ_{user} to compute hash collision value r' based on partial decryption result, and then generates signature σ'_{user} of the modified transaction.

3.3 Security Model

We first briefly describe the relevant security assumptions. TO is honest and will correctly add transactions to the blockchain. TMs with trusted devices might collude to collect credentials required for decryption. Authorities might be corrupted. PS might attempt to gather extra information in the decryption stage, while the correctness of the partial decryption is not affected. In this paper, we consider the following security guarantees.

Replayable Chosen Ciphertext Attack (RCCA) Security. In this paper, we adopt RCCA security [6] which allows the ciphertext to be modified, provided that the fundamental message cannot be adjusted explicitly. To demonstrate the security, we define the following security games between adversary $\mathcal{A}$ and challenger $\mathcal{C}$.

- *Setup*. The challenger $\mathcal{C}$ executes *Global_Setup* algorithm and sends public parameters to $\mathcal{A}$.
- *Key Query Phase I*. $\mathcal{C}$ initializes an integer $j = 0$, an empty set D, and an empty table T. $\mathcal{A}$ could adaptively and repeatedly execute any following queries:

  - *Create(S)*: $\mathcal{C}$ sets $j = j + 1$. For an attribute set S, $\mathcal{C}$ computes rewriting secret key and transformation key (SK, TK) by *Rew_KeyGen* and *Trans_KeyGen* algorithms, where $SK = \{SK_{i,gid}\}_{i \in S}$, $TK = \{TK_{i,gid}\}_{i \in S}$. Then $\mathcal{C}$ stores entry (j, S, SK, TK) into table T and sends TK to adversary $\mathcal{A}$.
  - *Corrupt(τ)*: If the τth entry exists in T, $\mathcal{C}$ retrieves this entry (τ, S, SK, TK) and stores S into D. Conversely, if the entry does not exist, it returns $\perp$.
  - *Decrypt(τ, CT)*: If T holds the τth entry, $\mathcal{C}$ queries entry (τ, S, SK, TK) and executes *Part_Decrypt* and *Full_Adapt* algorithms, and subsequently sends results to $\mathcal{A}$; otherwise, it returns $\perp$.
- *Challenge*. The adversary $\mathcal{A}$ sends to $\mathcal{C}$ two equal-length messages m_0, m_1. Meanwhile, $\mathcal{A}$ presents a challenge access policy $\mathbb{A}^* = (A^*, \rho^*)$ such that all attribute in D does not satisfy $\mathbb{A}^*$. Then $\mathcal{C}$ randomly sets a bit $b \in \{0, 1\}$ and executes *Offline_Hash* and *Online_Hash* algorithms on message m_b to compute a challenge ciphertext $CT^*_{m_b}$, and subsequently sends $CT^*_{m_b}$ to $\mathcal{A}$.
- *Key Query Phase II*. Repeating the phase I under following constraints:
  1) $\mathcal{A}$ cannot acquire the key that meets with $\mathbb{A}^*$.
  2) The message cannot be either m_0 or m_1 when $\mathcal{A}$ executes decryption query.
- *Guess*. The adversary $\mathcal{A}$ outputs a guess b' for b.

Definition 1. *Our scheme satisfies RCCA security if a probabilistic polynomial-time (PPT) adversary win the security game with a negligible advantage ε:*

$$\mathsf{Adv}^{\mathtt{RCCA}}_{\mathcal{A}} = |\Pr[b' = b] - 1/2| \leq \varepsilon. \tag{1}$$

Existential Unforgeability Under Chosen-Message Attacks (EUF-CMA). In this section, we introduce the existential unforgeability of BLS signature. We follow the definition of EUF-CMA in [4], and illustrate the security of the signature scheme against EUF-CMA by the following security game:

- *Setup*. $\mathcal{C}$ computes key pair (c, a) by key generation algorithm *Ken*, and then sends the public key c to $\mathcal{A}$.
- *Query Phase*. $\mathcal{A}$ queries for signature with respect to message m. $\mathcal{C}$ execute *Sign* algorithm to acquire signature σ and sends it to $\mathcal{A}$.
- *Output*. $\mathcal{A}$ outputs (m^*, σ^*). If $\mathsf{Ver}(c, m^*, \sigma^*) = 1$, m^* is absent from the query phase, then $\mathcal{A}$ wins this security game.

Let $\mathcal{S}$ be the signing oracle which takes any public key c and message m as the input, and outputs a signature σ satisfies $\mathsf{Ver}(c, m, \sigma)$. Given access to $\mathcal{S}$, the advantage of an adversary $\mathcal{A}$ is denoted as $\mathsf{Adv}^{\mathtt{EUF-CMA}}_{\mathcal{A}}$.

Definition 2. *$\mathcal{A}$ makes at most q_H and q_S queries to the hash function and signing oracle $\mathcal{S}$, respectively, up to t time. If $\mathsf{Adv}^{\mathtt{EUF-CMA}}_{\mathcal{A}}$ is negligible, then the signature scheme is EUF-CMA secure.*

Collision Resistance. A collisions for a chameleon hash can be found by an adversary $\mathcal{A}$ if her secret key satisfies the policy embedded in that hash. The interactions between $\mathcal{A}$ and challenger $\mathcal{C}$ are listed below:

- *Setup.* $\mathcal{C}$ executes *Auth_Setup* to generate key pair and sends public key to $\mathcal{A}$, and subsequently initializes an empty table Q, an integer $j = 0$ and a message space $\mathcal{M}$.
- *Query.* $\mathcal{A}$ executes key generation and adaption queries, and then acquires transformation key $TK = \{TK_{i,gid}\}_{\forall i}$ and collision $(m^*, h^*, r^*, \sigma^*_{\text{user}}, m'^*, r'^*, \sigma'^*_{\text{user}})$, where $(\sigma^*_{\text{user}}, \sigma'^*_{\text{user}})$ is signature. The transformation key TK and collision are recorded in Q.
- *Challenge.* $\mathcal{A}$ computes the hash collision. If the following equation holds: $\mathit{Hash\_Verify}(m^*, h^*, r^*, \sigma^*_{\text{user}}) = \mathit{Hash\_Verify}(m'^*, h^*, r'^*, \sigma'^*_{\text{user}})$, it returns 1; else returns 0.

The advantage of $\mathcal{A}$ is defined as follows:

$$\mathsf{Adv}^{\mathtt{CR}}_{\mathcal{A}} = \Pr[\mathcal{A} \to 1] \tag{2}$$

Definition 3. *If* $\mathsf{Adv}^{\mathtt{CR}}_{\mathcal{A}}$ *is negligible for any PPT adversaries* $\mathcal{A}$*, our scheme is collision resistance.*

Indistinguishability. Generally, indistinguishability implies that the adversary cannot distinguish whether the randomness of a chameleon hash is generated by the *Hash* algorithm or the *Adapt* algorithm. We define the following security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

- *Setup.* $\mathcal{C}$ executes the *Auth_Setup* algorithm and sends public key to $\mathcal{A}$.
- *Query Phase.* $\mathcal{C}$ selects a bit $b \in \{0, 1\}$ randomly. $\mathcal{A}$ executes $\mathsf{HashOrAdapt}$ queries $\mathcal{O}_{\mathtt{HashOrAdapt}}(.,.,.,.,.,.)$, which takes global parameter GP, public key pk_{AA}, messages m, m', an access policy $\mathbb{A}$ and $TK = \{TK_{i,gid}\}_{\forall i}$ as the input. $\mathcal{C}$ runs *Offline_Hash* and *Online_Hash* algorithms and obtains $(h_b, r_b, \sigma_{\text{user}b})$, and subsequently returns them to $\mathcal{A}$.
- *Guess.* $\mathcal{A}$ outputs its guess b'.

The advantage of $\mathcal{A}$ in the security game is defined as:

$$\mathsf{Adv}^{\mathtt{IND}}_{\mathcal{A}} = |\Pr[b = b'] - 1/2| \tag{3}$$

Definition 4. *For all PPT adversaries* $\mathcal{A}$*, our scheme satisfies indistinguishability if* $\mathsf{Adv}^{\mathtt{IND}}_{\mathcal{A}}$ *is negligible.*

4 Instantiation

4.1 Construction of Our Scheme

The ***Setup*** **phase** consists of the following algorithms:

- *Global_Setup*$(\lambda) \rightarrow GP$. AA generates global parameters $GP = (\mathbb{G}_1, \mathbb{G}_2, p, e)$ by this algorithm, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are bilinear groups of prime order p, e is a bilinear map $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. The generator g of $\mathbb{G}_1$ is chosen. In addition, a hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ is determined, which projects global identities gid to elements of $\mathbb{G}_1$. H_1 is regarded as a random oracle.
- *Auth_Setup*$(GP) \rightarrow (pk_{\text{AA}}, sk_{\text{AA}}, pk_{\text{sig}}, sk_{\text{sig}})$. Each authority is assumed to be responsible for one attribute in our scheme. Each AA owning attribute i chooses three exponents $\alpha_i, y_i, x_i \in \mathbb{Z}_p$ and computes its public key $pk_{\text{AA}} = \{e(g,g)^{\alpha_i}, g^{y_i}\}_{\forall i}$ and secret key $sk_{\text{AA}} = \{\alpha_i, y_i\}_{\forall i}$. It keeps ($pk_{\text{sig}} = \{g^{x_i}\}_{\forall i}, sk_{\text{sig}} = \{x_i\}_{\forall i}$ as the key pair for signature.
- *Modi_Setup*$(GP) \rightarrow (spk, ssk)$. TM randomly selects $z_m \in \mathbb{Z}_p$ and computes ($spk = g^{z_m}$, $ssk = z_m$) as its key pair.

The ***KeyGen*** **phase** is comprised of the following two algorithms:

- *Rew_KeyGen*$(GP, gid, i, sk_{\text{AA}}, sk_{\text{sig}}) \rightarrow (SK_{i,gid}, \sigma_{i,gid})$. For an authority owning attribute i, it executes operations listed below:
  1) Computing $SK_{i,gid} = g^{\alpha_i} H_1(gid)^{y_i}$ as the rewriting secret key for gid.
  2) Generating BLS signature $\sigma_{i,gid} = H_1(gid)^{x_i}$ for gid, and sending $SK_{i,gid}$ and $\sigma_{i,gid}$ to TM.
- *Trans_KeyGen*$(GP, SK_{i,gid}) \rightarrow TK_{i,gid}$. TM selects a randomness $z \in \mathbb{Z}_p$ and computes transformation key $TK_{i,gid} = (SK_{i,gid}^{1/z}, H_1(gid)^{1/z})$ for attribute i.

The ***Hash*** **phase** consists of the following two algorithms:

- *Offline_Hash*$(GP) \rightarrow (IC, IS)$. For attribute j, TO randomly selects exponents $\lambda'_j, \alpha'_j, y'_j, \omega'_j, r_j \in \mathbb{Z}_p$, then computes

$$C'_{1j} = e(g,g)^{\lambda'_j} \cdot e(g,g)^{\alpha'_j r_j};\ C'_{2j} = g^{r_j}; C'_{3j} = g^{y'_j r_j} g^{\omega'_j};$$
$$CT_{1j} = e(g,g)^{\alpha_j r_j} \cdot e(g,g)^{-\alpha'_j r_j};\ CT_{2j} = g^{y_j r_j} g^{-y'_j r_j}.$$

  Finally, $\{C'_{1j}, C'_{2j}, C'_{3j}\}_{\forall j}$ is reported as intermediate ciphertext IC, and $\{CT_{1j}, CT_{2j}\}_{\forall j}$ is regarded as intermediate state IS.
- *Online_Hash*$(GP, \{pk_{\text{AA}}\}, IC, IS, \mathbb{A}, m) \rightarrow (h, r, \sigma_{\text{user}})$. TO executes the following operations:
  1) TO defines a hash function $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p$, then selects a randomness $r \in \mathbb{Z}_p^*$ and a trapdoor T, and finally computes $h' = g^m \cdot p_{CH}^r$, where $p_{CH} = g^{H_2(T)}$.
  2) TO publishes ciphertext $CT = (\mathbb{A}, C_0, C'_0, IC, \{C'_{4j}, C'_{5j}\}_{\forall j}, IS)$. $\mathbb{A}$ is an $n \times l$ access matrix A with ρ mapping its rows to attributes. Given a security hash function H_3, TO chooses a randomness $R \in \mathbb{G}_2$, and

computes $s = H_2(R, T)$, $u = H_3(R)$. Then TO selects randomly a vector $v \in \mathbb{Z}_p^l$ where s is the first entry of v and a vector $w \in \mathbb{Z}_p^l$ where 0 is the first entry of w. Let λ_j denotes $A_j \cdot v$, ω_j denotes $A_j \cdot w$, and A_j is the jth row of A. The ciphertext is computed as:

$$C_0 = R \cdot e(g,g)^s;\ C_0' = T \oplus u;\ C_{4j}' = \lambda_j - \lambda_j';\ C_{5j}' = \omega_j - \omega_j'.$$

3) TO owns a signing key pair (spk', ssk'), where $spk' = g^{z_0} \in \mathbb{G}_1$ and $ssk' = z_0 \in \mathbb{Z}_p$. It computes signature $\sigma_{\text{user}} = H_1(h' \parallel \bar{r})^{ssk'}$, where $\bar{r} = g^{T+ssk'}$ denotes signed content.

Finally, TO outputs $(h, r, \sigma_{\text{user}}) = ((h', p_{CH}, CT), r, \sigma_{\text{user}})$, i.e. the hash, randomness and signature.

The ***Verify*** **phase** consists of the following two algorithms:

- *Aggre_Verify*$(gid, \{\sigma_{i,gid}\}, pk_{\text{sig}}) \rightarrow 0$ or 1. For reducing the cost of trial-and-error, The validity of rewriting secret key should be verified. TM aggregates the signatures $\{\sigma_{i,gid}\}$ from multiple authorities owning her attributes into one signature $\sigma_{gid} = \prod_i \sigma_{i,gid}$. Similarly, TM aggregates the signing public keys into $PK_{\text{sig}} = \prod_i (g^{x_i})$. The algorithm returns 1 if $e(\sigma_{gid}, g) = e(H_1(gid), PK_{\text{sig}})$, and returns 0 otherwise.
- *Hash_Verify*$(m, h, r, \sigma_{\text{user}}) \rightarrow 0$ or 1. Any entity could verify the transaction content on blockchain by checking whether all of the following satisfy: $h' = g^m \cdot p_{CH}^r$, $e(\sigma_{\text{user}}, g) = e(H_1(h' \parallel \bar{r}), spk')$. It returns 1 if all of them hold, and returns 0 otherwise.

The ***Adapt*** **phase** consists of the following two algorithms:

- *Part_Decrypt*$(GP, CT, \{TK_{i,gid}\}) \rightarrow CT'$. PS takes GP, ciphertext CT encrypted under access matrix $\mathbb{A} = (A, \rho)$ and transformation key $\{TK_{i,gid}\}$ as the input. For A_j (jth row of A), PS computes $C_{1j} = C_{1j}' \cdot CT_{1j} \cdot e(g,g)^{C_{4j}'}$, $C_{2j} = C_{2j}'$, $C_{3j} = C_{3j}' \cdot CT_{2j} \cdot g^{C_{5j}'}$, then chooses constant $c_j \in \mathbb{Z}_p$ such that $\sum_{j=1}^n c_j A_j = (1, 0, ..., 0)$, and computes CT_1, CT_2 as:

$$CT_1 = \prod_j \left(\frac{e(H_1(gid)^{\frac{1}{z}}, C_{3j})}{e(TK_{j,gid}^{\frac{1}{z}}, C_{2j})} \right)^{c_j} = \prod_j \left(\frac{e(H_1(gid), g)^{\frac{\omega_j}{z}}}{e(g,g)^{\frac{r_j \alpha_j}{z}}} \right)^{c_j},$$

$$CT_2 = \prod_j (C_{1j})^{c_j} = \prod_j \left(e(g,g)^{\lambda_j} e(g,g)^{\alpha_j r_j} \right)^{c_j}.$$

The partial decryption result $CT' = (C_0, C_0', CT_1, CT_2)$.

- *Full_Adapt*$(h, r, m, m', \{TK_{i,gid}\}, \sigma_{\text{user}}) \rightarrow (r', \sigma'_{\text{user}})$. The modifier holding transformation key $\{TK_{i,gid}\}$ executes the following operations to rewrite message m to be m':

1) Verify the transaction content as described above (Sect. 4.1).

2) Decrypt trapdoor T. TM computes $R = C_0/(CT_1 \cdot CT_2^{\frac{1}{z}})^z$ and $u = H_3(R)$. Subsequently, TM computes $T = C_0' \oplus u$ and $s = H_2(R, T)$. If $C_0 = R \cdot e(g,g)^s$ and $CT_1 \cdot CT_2^{\frac{1}{z}} = e(g,g)^{\frac{s}{z}}$, the trapdoor T is decrypted successfully; Otherwise, it outputs $\perp$.
3) Compute hash collision $r' = (m - m')/H_2(T)$.
4) Generate new signature $\sigma'_{\text{user}} = H_1(h' \parallel \bar{r}')^{ssk}$ for rewritten transaction, where $\bar{r}' = g^{T+ssk}$.

Finally, TM completes the rewriting operation and returns $(r', \sigma'_{\text{user}})$.

4.2 Correctness Analysis

In this section, we analyze whether the chameleon trapdoor T can be correctly calculated when TM's rewriting secret key is valid and attributes satisfy the access policy. Firstly, in the *Part_Decrypt* algorithm, we have:

$$C_{1j} = C'_{1j} \cdot CT_{1j} \cdot e(g,g)^{C'_{4j}} = e(g,g)^{\lambda'_j} e(g,g)^{\alpha'_j r_j} e(g,g)^{\alpha_j r_j} e(g,g)^{-\alpha'_j r_j} e(g,g)^{\lambda_j - \lambda'_j}$$
$$= e(g,g)^{\alpha_j r_j} e(g,g)^{\lambda_j};$$
$$C_{2j} = C'_{2j} = g^{r_j}; C_{3j} = C'_{3j} \cdot CT_{2j} \cdot g^{C'_{5j}} = g^{y'_j r_j} g^{\omega'_j} g^{y_j r_j} g^{-y'_j r_j} g^{\omega_j - \omega'_j} = g^{y_j r_j} g^{\omega_j};$$
$$CT_1 = \prod_j \left(\frac{e(H_1(gid), g)^{\frac{\omega_j}{z}}}{e(g,g)^{\frac{r_j \alpha_j}{z}}} \right)^{c_j} = \frac{1}{e(g,g)^{\sum \frac{c_j r_j \alpha_j}{z}}};$$
$$CT_2 = \prod_j (C_{1j})^{c_j} = \prod_j \left(e(g,g)^{\alpha_j r_j} e(g,g)^{\lambda_j} \right)^{c_j} = e(g,g)^s e(g,g)^{\sum \alpha_j r_j c_j}.$$

Secondly, in the *Full_Adapt* algorithm, the correct R can be calculated by substituting $C_0 = R \cdot e(g,g)^s$ into the following equation:

$$\frac{C_0}{\left(CT_1 \cdot CT_2^{\frac{1}{z}}\right)^z} = \frac{R \cdot e(g,g)^s}{\left(e(g,g)^{\sum \frac{c_j r_j \alpha_j}{z}} \left(e(g,g)^s e(g,g)^{\sum \alpha_j r_j c_j}\right)^{\frac{1}{z}}\right)^z} = \frac{R \cdot e(g,g)^s}{\left(e(g,g)^{\frac{s}{z}}\right)^z} = R.$$

Finally, the trapdoor T could therefore be calculated as: $T = C_0' \oplus H_3(R)$.

4.3 Security Proof

In this section, we provide the security proof of the proposed scheme. The proofs of collision resistance and indistinguishability were established in [9], and the security proof of EUF-CMA was shown in [4]. Therefore, only the proof of the RCCA security is given here.

Theorem 1. *If the construction of Lewko-Waters (LW) scheme [18] is selectively CPA security, then in the random oracle model, our proposed scheme is RCCA security regarding Definition 1.*

Proof. Suppose there exist an adversary $\mathcal{A}$ that can attack our scheme with advantage ε for any probabilistic polynomial-time (PPT). We subsequently build a simulator $\mathcal{B}$ that could successfully compromise the selective CPA security of LW scheme with advantage slightly less than ε.

- *Init.* $\mathcal{B}$ runs $\mathcal{A}$. $\mathcal{A}$ selects a challenge $\mathbb{A}^*$ and sends it to $\mathcal{B}$. $\mathcal{B}$ transmits this to the challenger of LW. We denote the LW challenger as $\mathcal{C}$.
- *Setup.* $\mathcal{B}$ computes the public parameters $PK = (e(g,g)^{\alpha_i}, g^{y_i})$ for all attributes i, and sends them to $\mathcal{A}$.
- *Phase I.* Then $\mathcal{B}$ initializes an empty set D, integer $j = 0$, and empty tables T_1, T_2, T_3. Subsequently, $\mathcal{B}$ answers the following queries from $\mathcal{A}$:
  - *Random Oracle Hash* $H_2(R,T)$: If there exists an entry (R,T,s) in T_1, returns s; otherwise, select $s \in \mathbb{Z}_p$, store (R,T,s) in T_1 and return s.
  - *Random Oracle Hash* $H_3(R)$: If there exists an entry (R,u) in T_2, returns u. Otherwise, select $u \in \{0,1\}^k$, store (R,u) in T_2 and return u.
  - *Create*(S): $\mathcal{B}$ sets $j = j+1$. If S does not satisfy $\mathbb{A}^*$, $\mathcal{B}$ executes the key generation algorithm to get $SK' = (PK, \{SK_{i,gid}\}_{i\in S})$. Then it selects $z \in Z_p$ and sets $TK = (PK, \{SK_{i,gid}^{1/z}\}_{i\in S})$ and $SK = (z, TK)$. Else, if S satisfies $\mathbb{A}^*$, $\mathcal{B}$ selects a randomness $d \in \mathbb{Z}_p$ and computes SK' by executing *Rew_KeyGen* algorithm to construct a fake transformation key. Subsequently, $\mathcal{B}$ sets $TK = SK'$, $SK = (d, TK)$ where TK is distributed appropriately for suitable selection of d. Finally, $\mathcal{B}$ stores (j, S, SK, TK) in T_3 and returns TK to $\mathcal{A}$.
  - *Corrupt(*i*).* If entry (i, S, SK, TK) exists, $\mathcal{B}$ can obtain it, set $D = D \cup S$ and return SK to $\mathcal{A}$.
  - *Decrypt(*i, CT*).* Suppose the ciphertext has been partially decrypted, both $\mathcal{A}$ and $\mathcal{B}$ could perform key transformation algorithm since they possesses transformation key TK. Let $CT = (C_0, C_0', CT_1, CT_2)$ related to $\mathbb{A}^*$. (i, S, SK, TK) is acquired if it exists in T_3, and $\perp$ is returned if not or $S \notin \mathbb{A}^*$. In addition, if key i does not satisfy $\mathbb{A}^*$, the following operations are performed:

1) Parse $SK = (z, TK)$, and calculate $R = C_0/(CT_1 \cdot CT_2^{1/z})^z$.
2) Obtain (R, T_i, s_i) from T_1 if exists; otherwise return $\perp$.
3) If $\exists y \neq x$ that satisfies (R, T_y, s_y) and (R, T_x, s_x) are presented in T_1, $T_y = T_x$ and $s_y = s_x$, the simulation is terminated.
4) Otherwise, retrieve (R, u) from T_2 if it exists. Else return $\perp$.
5) Verify if $C_0 = R \cdot e(g,g)^{s_i}$, $C_0' = T_i \oplus u$, $CT_1 \cdot CT_2^{1/z} = e(g,g)^{s_i/z}$ for each i.
6) If attribute i exists and could pass the aforementioned checking, output T_i, else $\perp$.

If there exists i that satisfies $\mathbb{A}^*$, then perform operations listed below:

1) Parse $SK = (d, TK)$, and calculate $\beta = (CT_1 \cdot CT_2^{1/d})^d$.
2) Check whether $\beta = e(g,g)^{s_i}$ for each entry (R, T_i, s_i) in T_1.
3) If no such entry exists, $\mathcal{B}$ returns $\perp$.
4) If the entry that meets the condition is not unique, $\mathcal{B}$ terminates the operation.
5) Otherwise, let (R, T, s) to be the only entry satisfied. Then obtain (R, u) from T_2 if it exists, otherwise output $\perp$.

6) $\mathcal{B}$ verifies if $C_0 = R \cdot e(g,g)^s$, $C_0' = T \oplus u$, $(CT_1 \cdot CT_2^{1/d})^d = e(g,g)^s$.
7) Output T if all conditions are met, else return $\perp$.

- *Challenge.* $\mathcal{A}$ sets two messages (T_0^*, T_1^*). $\mathcal{B}$ randomly selects messages $(R_0, R_1) \in G_2^2$, then calls $\mathcal{C}$ to get ciphertext $CT = (C_0, \{C_{1j}, C_{2j}, C_{3j}\}_{\forall j})$ with $\mathbb{A}^*$. $\mathcal{B}$ randomly selects $C_0' \in \{0,1\}^k$ and sends $\mathcal{A}$ the challenge ciphertext $CT^* = (C_0, C_0', \{C_{1j}, C_{2j}, C_{3j}\}_{\forall j})$.
- *Phase II.* $\mathcal{B}$ performs the same answers operation as in Phase I except that the decryption query is either T_0^* or T_1^*, then produces the test message.
- *Guess.* $\mathcal{A}$ should output a bit otherwise terminate the operation, while $\mathcal{B}$ would ignores it in any cases. $\mathcal{B}$ examines if any entry in T_1 and T_2 contains R_0 or R_1 as its first element. If neither randomness meets the condition, $\mathcal{B}$ returns its guess in $\{0,1\}$. If only R_b exists, $\mathcal{B}$ outputs b. If a correct guess is proposed by $\mathcal{A}$, it implies that $\mathcal{A}$ is aware of R_b with probability ε and R_b is retrieved through H_1 or H_2 oracle with ε. Then, $\mathcal{B}$ could produce a correct guess with the probability slightly larger than ε.

Therefore, if $\mathcal{A}$ could break our scheme with the given advantage ε, then with the same advantage, $\mathcal{B}$ could break the LW scheme [18]. Hence, the Theorem 1.

5 Performance Analysis

In this section, we evaluate the performance of our proposed scheme from the perspective of functionality comparison and computational burden.

Functionality Comparison. Table 1 provides the functionality comparison among our proposed scheme and several related rewritable blockchain schemes [9,10,14,28,33]. It is seen that our scheme is the only one that satisfies all of the properties, i.e. decentralized rewriting, fine-grained access control, offline/online hash, outsourced computation, and bi-level validity verification. Centralized attribute authority is not required in our scheme, instead, multiple attribute authorities share the responsibility for the distribution of the rewriting secret key. The schemes in [9,14,28] cannot support decentralized rewriting. The scheme in [14] makes use of multiple attribute authorities, nevertheless, it still requires

Table 1. Functionality comparison

References	[10]	[9]	[28]	[33]	[14]	Ours
Decentralized rewriting	✓	×	×	✓	×	✓
Fine-grained access control	×	✓	✓	✓	✓	✓
Offline/Online hash	–	×	×	×	✓	✓
Outsourced computation	–	×	×	×	✓	✓
Bi-level validity verification	–	×	×	×	×	✓

"✓":Well-done; "×":Not achieved; "–":Considered but needs further implements.

authorities to negotiate and share the same private key, implying that decentralization is not implemented in essence. The scheme in [10] cannot implement fine-grained access control. The scheme in [33] cannot perform offline/online hash and outsourced computation, as a result, users with resource-constrained devices may struggle to use in practice. Moreover, none of the above schemes have implemented the bi-level verification in blockchain rewriting.

Computational Burden. We implement our scheme in Python 3.8 and the Charm framework [1] on a workstation with Intel Xeon(R) E5-1620v4 CPU 3.50GHz and 128GB RAM. We adopt Type A curve from Pairing-Based Cryptography library for pairing, which has base field size of 512 bits. Specifically, we measure the running times of five major algorithms.

For comprehensive and fair comparison, we increase the attribute size from 10 to 100, and the corresponding access policy is set in the form of "(S_1 and S_2) or (S_3 and S_4) or...", where S_i denotes an attribute. Each instance is run 100 times to estimate the average running time.

As illustrated in Fig. 2, the running times are basically invariant with the numbers of attributes in *Setup*/*Global_Setup*[1] algorithms for each scheme, while our scheme achieves better performance. We conclude the reason is that our scheme dispenses with complex operations such as generating master keys during the setup phase, due to the decentralized design. Figure 3 shows that the running times of *KeyGen*/*Rew_KeyGen* algorithms grow approximately linear as attributes increasing in each scheme. The running time of our scheme is acceptable, which is less than most other schemes and only slightly higher than scheme in [9]. Figure 4 illustrates the running times of *Hash*/*Online_Hash* algorithms with the size of policies increasing. It is seen that the running time of our scheme is always kept to the lowest, while is present as increasing functions of policies for other schemes. It is due to the fact that the majority of computations (e.g. computation of IC and IS) are executed offline, which reduces

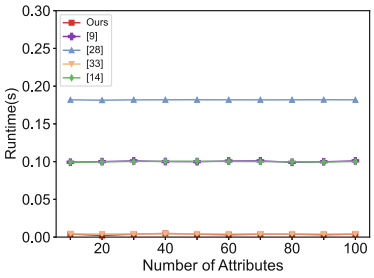

Fig. 2. Running time of *Setup*/*Global_Setup*

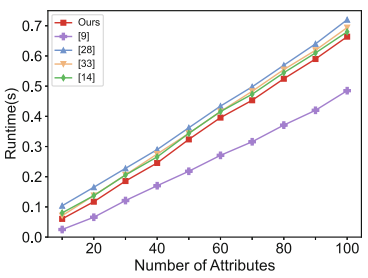

Fig. 3. Running time of *KeyGen*/*Rew_KeyGen*

[1] Algorithms separated by slashes represent functionally identical stages in various schemes, albeit with different names.

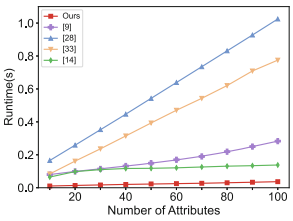

Fig. 4. Running time of *Hash*/*Online_Hash*

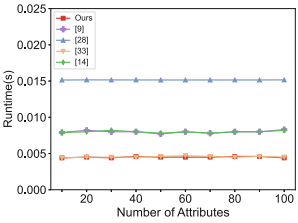

Fig. 5. Running time of *Verify*/*Hash_Verify*

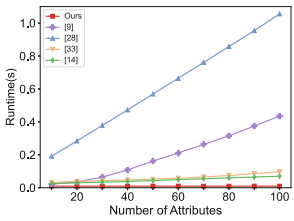

Fig. 6. Running time of *Adapt*/*Full_Adapt*

the online computation cost. Likewise, as demonstrated in Fig. 5, our scheme and [33] achieve better performance in *Verify*/*Hash_Verify* algorithm. It takes 0.004 s to execute the algorithm with 100 attributes for our scheme, which is 0.011 s less than [28]. Moreover, our scheme achieves satisfactory running time in *Adapt*/*Full_Adapt* algorithm with increasing size of polices (shown in Fig 6). For reducing the computation burden of users, the majority of the computations in *Adapt* phase are separated out and executed by the proxy sever (partial decryption) in our scheme. As a result, with increasing size of policies, the running time of our scheme remains stably at the lowest, whereas of the schemes in [9,28] grow appreciably.

6 Conclusion

In this paper, we proposed a scheme of decentralized and efficient blockchain rewriting with bi-level validity verification. The scheme overcomes the limitation of requiring a single trusted authority for the distribution of modification permissions compared to previous schemes, and supports multiple authorities to jointly manage and distribute the rewriting secret key. Meanwhile, the arrangement that separating the hash computation and the decryption into two stages can significantly reduce the computational burden on the user side, which is more conducive to the application in practice. Furthermore, the scheme supports validity verification for the modifier to check the validity of rewriting secret keys, as well for any entity to verify the content on blockchain. To the best of our knowledge, our scheme is the first to simultaneously support properties of decentralization, offline/online hash, outsourced computation and bi-level validity verification. Finally, extensive experimental results demonstrate that our scheme is advantageous in both functionality and performance.

Acknowledgements. We thank the anonymous reviewers for the valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (No. 62072359, No. 62072352, No. 61902292).

References

1. Akinyele, J.A., et al.: Charm: a framework for rapidly prototyping cryptosystems. J. Cryptographic Eng. **3**(2), 111–128 (2013)
2. Ateniese, G., Magri, B., Venturi, D., Andrade, E.: Redactable blockchain-or-rewriting history in bitcoin and friends. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 111–126. IEEE (2017)
3. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: 2007 IEEE Symposium on Security and Privacy (S&P 2007), 20–23 May 2007, Oakland, California, USA, pp. 321–334. IEEE Computer Society (2007)
4. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. J. Cryptology **17**(4), 297–319 (2004)
5. Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Chameleon-hashes with ephemeral trapdoors. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10175, pp. 152–182. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54388-7_6
6. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_33
7. Chase, M.: Multi-authority attribute based encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 515–534. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_28
8. De Aguiar, E.J., Faiçal, B.S., Krishnamachari, B., Ueyama, J.: A survey of blockchain-based strategies for healthcare. ACM Comput. Surv. (CSUR) **53**(2), 1–27 (2020)
9. Derler, D., Samelin, K., Slamanig, D., Striecks, C.: Fine-grained and controlled rewriting in blockchains: chameleon-hashing gone attribute-based. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, 24–27 February 2019. The Internet Society (2019)
10. Deuber, D., Magri, B., Thyagarajan, S.A.K.: Redactable blockchain in the permissionless setting. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 124–138. IEEE (2019)
11. Dutta, P., Choi, T.M., Somani, S., Butala, R.: Blockchain technology in supply chain operations: applications, challenges and research opportunities. Transp. Res. Part E: Logist. Transp. Rev. **142**, 102067 (2020)
12. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006, pp. 89–98. ACM (2006)
13. Guo, F., Mu, Y., Chen, Z.: Identity-based online/Offline encryption. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 247–261. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_22
14. Guo, L., Wang, Q., Yau, W.-C.: Online/offline rewritable blockchain with auditable outsourced computation. IEEE Trans. Cloud Comput., 1 (2021). https://doi.org/10.1109/TCC.2021.3102031
15. Hohenberger, S., Waters, B.: Online/Offline attribute-based encryption. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 293–310. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54631-0_17

16. Krawczyk, H., Rabin, T.: Chameleon signatures. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA (2000)
17. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: attribute-based encryption and (Hierarchical) inner product encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_4
18. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 568–588. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_31
19. Lewko, A., Waters, B.: New proof methods for attribute-based encryption: achieving full security through selective techniques. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 180–198. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_12
20. Lewko, A.B., Waters, B.: Decentralizing attribute-based encryption. In: Proceedings of Advances in Cryptology - EUROCRYPT 2011–30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, 15–19 May 2011, vol. 6632, pp. 568–588 (2011)
21. Li, J., Zhang, Y., Ning, J., Huang, X., Poh, G.S., Wang, D.: Attribute based encryption with privacy protection and accountability for cloudiot. IEEE Trans. Cloud Comput. **10**, 762–773 (2020)
22. Maram, S.K.D., et al.: Churp: dynamic-committee proactive secret sharing. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2369–2386 (2019)
23. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Bus. Rev. 21260 (2008)
24. Puddu, I., Dmitrienko, A., Capkun, S.: μchain: How to forget without hard forks. Cryptology ePrint Archive (2017)
25. Qi, S., Lu, Y., Zheng, Y., Li, Y., Chen, X.: Cpds: enabling compressed and private data sharing for industrial internet of things over blockchain. IEEE Trans. Ind. Inf. **17**(4), 2376–2387 (2020)
26. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_27
27. Thyagarajan, S.A.K., Bhat, A., Magri, B., Tschudi, D., Kate, A.: Reparo: publicly verifiable layer to repair blockchains. In: Borisov, N., Diaz, C. (eds.) FC 2021. LNCS, vol. 12675, pp. 37–56. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64331-0_2
28. Tian, Y., Li, N., Li, Y., Szalachowski, P., Zhou, J.: Policy-based chameleon hash for blockchain rewriting with black-box accountability. In: Annual Computer Security Applications Conference, pp. 813–828 (2020)
29. Tian, Y., Liu, B., Li, Y., Szalachowski, P., Zhou, J.: Accountable fine-grained blockchain rewriting in the permissionless setting. arXiv preprint arXiv:2104.13543 (2021)
30. Voigt, P., von dem Bussche, A.: The EU General Data Protection Regulation (GDPR), vol. 1. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57959-7
31. Xie, M., Ruan, Y., Hong, H., Shao, J.: A CP-ABE scheme based on multi-authority in hybrid clouds for mobile devices. Future Gener. Comput. Syst. **121**, 114–122 (2021)

32. Yu, Y., Guo, L., Liu, S., Zheng, J., Wang, H.: Privacy protection scheme based on CP-ABE in crowdsourcing-IoT for smart ocean. IEEE Internet Things J. **7**(10), 10061–10071 (2020)
33. Zhang, Z., Li, T., Wang, Z., Liu, J.: Redactable transactions in consortium blockchain: controlled by multi-authority CP-ABE. In: Baek, J., Ruj, S. (eds.) ACISP 2021. LNCS, vol. 13083, pp. 408–429. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90567-5_21