


Article

Policy-Based Chameleon Hash with Black-Box Traceability for Redactable Blockchain in IoT

Pengfei Duan , Jingyu Wang, Yuqing Zhang, Zhaofeng Ma and Shoushan Luo

School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China

* Correspondence: pengfeiduan@bupt.edu.cn

Abstract: Blockchain has become an integral part of various IoT applications, and it has been successful in boosting performance in various aspects. Applying blockchain as a trust solution for Internet-of-Things is a viable approach. The immutability of blockchain is essential to prevent anyone from manipulating registered IoT data transactions for illegitimate benefits. However, the increasing abuse of blockchain storage negatively impacts the development of IoT blockchain and potential stakeholders are discouraged from joining the IoT data sharing as the IoT data recorded on the blockchain contains private information. Hence, it is crucial to find ways to redact data stored on the IoT blockchain, which is also supported by relevant laws and regulations. Policy-based chameleon hash is useful primitive for blockchain rewriting, allowing the modifier to rewrite the transaction if they possess enough rewriting privileges that satisfy the access policy. However, this approach lacks traceability, which can be exploited by malicious modifiers to grant unauthorized user modification privileges for personal gain. To overcome this deficiency, we introduce a new design of policy-based chameleon hash with black-box traceability (PCHT), which enables the authority to identify the set of producers responsible for generating the pirate decoder. Specifically, PCHT is constructed by practical attribute-based encryption with black-box traceability (ABET) and collision-resistant chameleon hash with ephemeral trapdoor (CHET). After modeling PCHT, we present its concrete instantiation and rigorous security proofs. Finally, a PCHT-based redactable transaction scheme for IoT blockchain is given. Compared to the state-of-the-art mutable blockchain solutions, our scheme provides fine-grained blockchain rewriting and black-box traceability. The evaluation results demonstrate that our scheme is efficient and practical while still ensuring that no computational overhead is placed on IoT devices with limited computing resources.

Keywords: IoT blockchain; blockchain rewriting; chameleon hash; black-box traceability; fingerprinting code



Citation: Duan, P.; Wang J.; Zhang Y.; Ma, Z.; Luo, S. Policy-Based Chameleon Hash with Black-Box Traceability for Redactable Blockchain in IoT. *Electronics* **2023**, *12*, 1646. <https://doi.org/10.3390/electronics12071646>

Academic Editor: Keke Gai

Received: 18 February 2023

Revised: 18 March 2023

Accepted: 20 March 2023

Published: 30 March 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

As a prevalent computing technology paradigm, the Internet of Things (IoT) is widely utilized in various fields, including smart healthcare [1], smart grids [2], and smart cities [3,4]. However, with the rapid increase in IoT applications, they are confronted with numerous security and privacy challenges. Common attack methods include data sniffing, data spoofing, and malicious data injection [5,6]. It is evident that the target of these attacks is the data stored on the IoT network [7]. Typically, a centralized database is used to process the collected IoT data. Nevertheless, this approach has centralized drawbacks, and blockchain offers a promising solution for achieving distributed data protection in IoT networks [8,9].

Blockchain has gained tremendous popularity and acceptance by a wider community since Satoshi Nakamoto proposed Bitcoin in 2008 [10]. As a decentralized ledger technology, blockchain can be an ideal solution for multicenter cooperation-based trust [11,12]. Nowadays, it has been widely applied in insurance [13,14], DRM [15,16], and IoT [17,18], not just focusing on cryptocurrencies. For blockchain, it is well known that blocks are linked by hash chain and immutability is one of its characteristics. Specifically, a block contains

multiple transactions. Generally, any modification on a transaction is nearly impossible since it would influence the continuity of the block.

With the progress of IoT technology, IoT data are now not confined to a singular closed environment, but is distributed amongst various cooperative IoT system [19]. In order to cater to intricate application requirements, sharing of data among different IoT systems is becoming an inevitable trend. For an IoT data owner, it is reasonable to process their own data, which is also supported by relevant laws and regulations. For example, the general data protection regulation (GDPR) [20] gives data owners the right of alerting their data. Additionally, the immutability of blockchain can lead to several challenges. On the one hand, the increasing abuse of blockchain storage negatively impacts the development of IoT blockchain. On the other hand, there is a risk of inappropriate content and sensitive information being permanently recorded on the blockchain, which may discourage potential stakeholders from joining IoT data sharing. Hence, it is crucial to find ways to erase data stored on the IoT blockchain.

To make data posted to a IoT blockchain redactable, there are several existing strategies, which can be categorized into two types: hard fork and chameleon hash. The former creates forked blockchain at the block where the data need to be changed and discards the subsequent blocks. However, this method cannot achieve fine-grained data modifying. The latter applies chameleon hash to achieve secure blockchain rewriting. Ateniese et al. proposed a seminal block-level rewriting scheme by replacing the traditional hash function with the chameleon hash, where the trapdoor owners could rewrite the block without breaking the link of each two adjacent blocks [21]. To make the modification more fine-grained and achieve transaction-level rewriting, Derler et al. introduce attribute-based encryption (ABE) into the chameleon hash called policy-based chameleon hash (PCH), in which a party could create a modifiable transaction using PCH and only the specific modifier who has the associated attributes could modify it. Compared to [21], here chameleon hashes are used to hash the transaction in computing the Merkle root. PCH is derived from CHET [22] and attribute-based encryption (ABE) [23]. Specifically, the long-term trapdoor in CHET is generated by the central authority and will be issued to the modifier and the ephemeral one is derived from the transaction owner for each transaction. The ephemeral trapdoor is sealed by the ABE scheme.

There are multiple recipients in the ABE scheme and each one is assigned an attribute-related key to modify the data recorded in a transaction. Generally, this key is not allowed to be shared with others. However, this key management policy is invalid for the malicious modifier. By constructing the pirate decoder with his decryption key, the malicious modifier can issue his modification privileges to the unauthorized and gain certain benefits. In this case, the holder of the pirate decoder can maliciously change shared data on the IoT blockchain, e.g., tampering with original sensor data, for personal gain, thereby distributing the normal operation of data modification on the IoT blockchain. Hence, it is necessary to trace the corrupted modifier and then enforce the correct punishments.

Several redactable blockchain schemes with traceability have been proposed. Tian et al. proposed the PCH-based redactable blockchain with accountability, which links the modified transaction to its modifier but not the key leakage [24]. Additionally, Panwar et al. introduced a permissioned redactable blockchain with traceability and revocability by combining dynamic group signature and FAME, but their solution only provides weak accountability that links the modified transaction to its modifier but not producers of the pirate decoder [25]. To achieve strong accountability, the traceability object in this paper is the collude producers, named traitors, of the pirate decoder that allows the unauthorized to rewrite transactions.

Based on ABET and CHET, policy-based chameleon hash with traceability, named PCHT, is introduced in this paper. For constructing a practical ABET with black-box traceability, we choose the efficient CP-ABE scheme [23] and robust fingerprinting code [26]. Moreover, PCHT has constant-size ciphertext. The major contributions are summarized as follows.

- Formal definition and security model. The formal definition and security model of policy-based chameleon hash with traceability (PCHT) are introduced. The remarkable feature of PCHT is that it enables the authority to trace the corrupt modifiers who produce the pirate decoder that allows the unauthorized to rewrite data recorded in IoT blockchain. Compared to the existing solutions, our solution achieves black-box traceability while equipping security and performance advantages.
- Generic construction and practical instantiation. We begin by providing a detailed explanation of the generic construction of PCHT. Following this, we present a PCHT-based redactable transaction scheme for IoT blockchain. Finally, we describe a practical instantiation of PCHT. Considering the limited computing resources of IoT devices, the data sharing is conducted by the data owner and the transaction rewriting operations are performed by full nodes on the IoT blockchain, which have sufficient computational resources to execute the hashing and adaptation algorithms. Due to the infrequent occurrence of transaction rewriting in the IoT system, it is not expected to significantly impact the system's performance in a negative way.

2. Overview

In the system model of PCHT-based redactable blockchain for IoT, there are three types of entities, which are shown in Figure 1. Then, we introduce the cryptographic primitives behind PCHT.

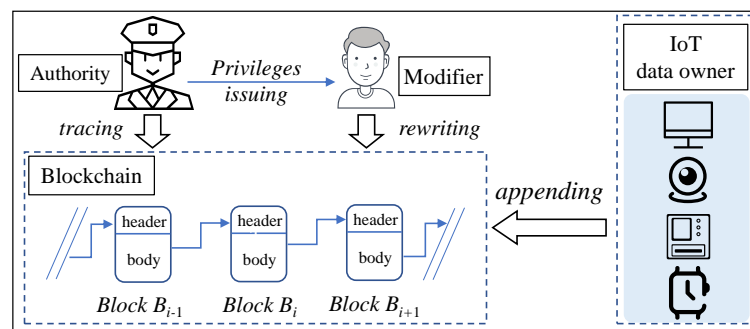


Figure 1. System model of PCHT-based redactable blockchain for IoT.

2.1. System Overview

First, we introduce a trusted authority to take charge of system initialization and traitor tracing. Generally, the transactions are published by the IoT data owner and redacted by the modifier. The blockchain data rewriting in blockchain for IoT could be performed by the following steps.

- System initialization. The authority initializes the system by publishing the public parameters and generating a secret key for each participant using their individual attribute sets.
- Mutable transaction publication. The IoT data owner appends a mutable transaction to the blockchain using the hash algorithm of PCHT. Then, they would check its validity by the verify algorithm of PCHT to decide whether to record it in the local ledger.
- Mutable transaction rewriting. When the attribute set of modifier satisfies the modification policy, he can rewrite this mutable transaction. After that, the modifier broadcasts it, and the other participants would also check its validity.
- Traitor tracing. If there exists a pirate decoder for modifying the mutable transactions illegally, the authority would trace the corrupted modifiers by the tracing algorithm of PCHT.

2.2. Cryptographic Building Blocks

For PCHT, CHET and ABET are two critical components.

- CHET. Blockchain rewriting is realized by CHET in this paper. There are two trapdoors in CHET: the long-term and the ephemeral. The former is generated by the authority in system initialization. The latter is computed by the IoT data owner during mutable transaction publication. Only if the modifier has both of them could he succeed in rewriting the mutable transaction.
- ABET. It is a key component to construct PCHT, which can achieve fine-grained blockchain rewriting and offer black-box traceability in traitor tracing. The ephemeral trapdoor encrypted with the encrypting algorithm of ABET could only be decrypted by the modifier who has the proper attributes. Compared with traditional CP-ABE schemes, ABET has an additional tracing algorithm that takes public parameters and tracing key as input and outputs the identity set of traitors.

2.3. Instantiation and Implementation

According to the generic construction scheme of ABET introduced in [27], we choose FAME [23] and robust fingerprinting code [26] to initiate it. In addition to this, we also rely on the discrete logarithm-based CHET [22]. Moreover, the PCHT-based redactable blockchain scheme for IoT is also built in the paper, which makes the pirate decoder-related corrupted modifiers traceable.

3. Related Work

Traceable CP-ABE. Generally, the tracing algorithm of traceable CP-ABE scheme has access to the pirate decoder D_S and takes tracing key tk as input, outputs the index $i \in \{1, \dots, k\}$ that indicates a secret key sk_i was used to create the pirate decoder. It could be divided into two categories: public tracing and secret tracing.

An augmented broadcast encryption with public traitor tracing was introduced in [28] and the tracing algorithm can be executed by any public user. However, it has sub-linear size ciphertext $\mathcal{O}(\sqrt{k})$. Later, Ning et al. proposed a traceable CP-ABE scheme with constant-size ciphertext $\mathcal{O}(1)$ where the construction is built on top of a CP-ABE [29] and an anonymous HIBE [30]. Although it has impressive practicality in commercial [31], this scheme only supports white-box traceability, i.e., its traceability property only works when the malicious users leak the well-formed decryption keys directly. An expressive black-box traceable CP-ABE was proposed in [32], while it has sub-linear size ciphertext $\mathcal{O}(\sqrt{k})$ so that the practicality is hindered seriously. Moreover, public tracing is not suitable for the tracing of blockchain rewriting since the blockchain rewriting event happens infrequently and pirate tracing is the responsibility for the authority in order to force the key management policy.

Secret tracing means the tracing algorithm treats tracing key tk as secret value and tk is only hold by the authority. Fingerprinting code is usually used to construct secret tracing scheme. Boneh and Naor proposed the public-key-based traitor tracing system with constant-size ciphertext $\mathcal{O}(1)$ [33]. The proposed system relies on a fingerprinting code [34]. The fingerprinting code includes a code generator algorithm and a tracing algorithm which is used for traitor tracing. Recently, Lai et al. introduced a traceable attribute-based encryption scheme with constant-size ciphertext, which is also based on the fingerprinting code [27]. The proposed construction is built on an efficient CP-ABE scheme [23], and a Tardos code [35]. In this paper, we construct a practical ABET scheme with black-box traceability, while supporting constant-size ciphertext $\mathcal{O}(1)$ and secret tracing. A difference comparison among the existing ABET schemes is shown in Table 1.

Table 1. Comparison of traceable CP-ABE schemes.

	CP-ABE Scheme	Ciphertext Size	Black-Box	Secret Tracing
[32]	ABE [29]	$\mathcal{O}(\sqrt{k})$	✓	×
[31]	ABE [29]	$\mathcal{O}(1)$	×	×
[24]	FAME [23]	$\mathcal{O}(1)$	✓	×
Ours	FAME [23]	$\mathcal{O}(1)$	✓	✓

Redactable blockchain. To achieve blockchain redacting, many schemes were proposed and they could be categorized into two types [36]. The non-cryptography-based schemes realize blockchain rewriting by some mechanisms. Puddu et al. proposed μ chain and realized blockchain redacting by a voting-like approach where each participant was assigned a key sharing by a dynamic proactive secret sharing scheme and the modifier had to collect enough votes to achieve blockchain redacting [37]. The cryptography-based schemes mainly adopted the chameleon hash to achieve blockchain redacting and a series of notable works have been proposed.

The chameleon hashing and signatures were put forward by Krawczyk and Rabin [38]. In 2017, Ateniese et al. first introduced the concept of redactable blockchain where the traditional hash function was replaced by a chameleon hash function to calculate the hash and only the party who owns the trapdoor could efficiently find valid collisions to rewrite the blockchain without changing the hash output [21]. To achieve blockchain redacting in a fine-grained and controlled way, Derler et al. proposed a novel cryptographic primitive, named policy-based chameleon hash (PCH), and introduced it into the blockchain redacting [39]. Anyone who satisfies the policy embedded in the transaction could then find arbitrary collisions for a given hash. The blockchain rewriting could be manipulated at a transaction level when PCH is applied to the blockchain. After that, To achieve rewriting authorization in a decentralized setting, Zhang et al. introduced a multi-authority policy-based chameleon hash (MPCH) [40], while Ma et al. presented a decentralized policy-based chameleon hash (DPCH) [41]. Both MPCH and DPCH utilize CHET to manage data rewriting, with multi-authority attribute-based encryption [42] being leveraged to handle the rewriting privilege. However, neither of them considered tracing the malicious modifiers.

Tian et al. were the first to consider accountability in PCH-based redactable blockchains. However, their solution only provides weak accountability by linking the modified transaction to its modifier without addressing the issue of key leakage [24]. Panwar et al. proposed a permissioned redactable blockchain that provides traceability and revocability through the use of dynamic group signature schemes (DGSS) and revocable FAME (RFAME). While their solution links modified transactions to their modifiers and offers revocation of malicious modification privileges, the traceability only offers weak accountability, and the revocation mechanism has a linear complexity with the size of non-revoked modifiers [25].

To overcome the limitations of current mutable blockchain solutions, we have proposed a PCHT-based blockchain rewriting scheme with black-box traceability, which enables traitor tracing and effectively resolves the issue of key leakage. In Table 2, we compare our scheme with existing mutable blockchain solutions.

Table 2. Comparison of mutable blockchain schemes.

	Underlying Method	Fine-Grained	Traceability	Traitor Tracing
[37]	hard fork	×	×	×
[39]	CHET	✓	×	×
[24]	CHET	✓	✓	×
Ours	CHET	✓	✓	✓

4. Preliminary

In this section, several relevant cryptography fundamentals involved in PCHT are presented and the important notations used in this paper are listed in Table 3.

Table 3. The notations used in this paper.

Notation	Definition
DO	the IoT data owner
TM	the transaction modifier
sk_{do}	the secret key of IoT data owner
sk_{tm}	the secret key of transaction modifier
\mathbb{A}	the access structure
M	the matrix representing a MSP
ρ	the mapping function
S	the set of attributes
T	the identity set of traitors
D_S	the pirate decoder for a set of attributes S

4.1. Chameleon Hash with Ephemeral Trapdoors

In CHET [22], the ephemeral trapdoor is chosen during hashing algorithm. It is necessary to hold both the long-term and the ephemeral trapdoor to computing hash collision successfully. This primitive includes the following five algorithms:

- $CHET.Setup(1^\lambda)$: It requires the security parameter λ as input, and outputs the public parameter pp .
- $CHET.KGen(pp)$: It requires pp as input, and outputs the key pair (pk_{CHET}, sk_{CHET}) where the sk_{CHET} is used as the long-term trapdoor.
- $CHET.Hash(pk_{CHET}, m)$: It requires pk_{CHET} and a message m as input, and outputs hash η , randomness r , and the ephemeral trapdoor etd .
- $CHET.Verify(pk_{CHET}, m, \eta, r)$: It requires pk_{CHET} , m , η , and r as input, and outputs a bit $b \in \{0, 1\}$.
- $CHET.Adapt(sk_{CHET}, etd, m, m', \eta, r)$: It requires sk_{CHET} , etd , m , m' , η , and r as input, and outputs the new randomness r' .

Correctness: Note that the verifying algorithm would always verify if the given hash is valid, and outputs \perp otherwise. Moreover, the correctness of CHET could be described as:
For all: $\lambda \in \mathbb{Z}$,

$$\begin{aligned} pp &\leftarrow CHET.Setup(1^\lambda), \\ (sk_{CHET}, pk_{CHET}) &\leftarrow CHET.KGen(pp), \\ (\eta, r, etd) &\leftarrow CHET.Hash(pk_{CHET}, m), \\ r' &\leftarrow CHET.Adapt(sk_{CHET}, etd, m, m', \eta, r), \end{aligned}$$

we have that:

$$CHET.Verify(pk_{CHET}, m, \eta, r) = CHET.Verify(pk_{CHET}, m', \eta, r') = 1.$$

For security, CHET is required to be indistinguishable, i.e., any adversary could not distinguish the randomness generated in hashing step or adapting step. Moreover, CHET also satisfies the collision-resistant (i.e., any adversary without holding both the long-term and the ephemeral trapdoor simultaneously cannot find any hash collision).

4.2. Monotone Span Program (MSP)

Specifically, let $\{x_1, x_2, \dots, x_n\}$ be a set of boolean variables, the MSP is a labeled matrix, denoted as $\tilde{M}(M, \rho)$ where ρ is a mapping $\rho: \{1, \dots, n_1\} \rightarrow \mathcal{U}$ and $\rho(i) = x_i$ (i.e., the i th row of M is marked as x_i).

Suppose $\delta \in \{0, 1\}^n$ is the input of boolean function f , the sub matrix \mathbf{M}_δ consists of d rows of \mathbf{M} that satisfies the following condition: the label of row is x_i and $\delta_i = 1$. We say that the (\mathbf{M}, ρ) accepts δ iff $\vec{1} \in \text{span}(\mathbf{M}_\delta)$ where $\vec{1} = (1, 1, \dots, 1)$ and $\text{span}(\mathbf{M}_\delta)$ is a certain linear combination of all rows in \mathbf{M}_δ .

The linear secret-sharing scheme could be built on MSP. First, suppose $\{P_1, P_2, \dots, P_n\}$ is the set of participants and for $G \in \{P_1, P_2, \dots, P_n\}$, $\delta_G \in \{0, 1\}^n$ is the feature vector of G , which means the i th coordinate is 1 only if $P_i \in G$. We consider the column vector $\vec{r} = (r_1, r_2, \dots, r_{n_2})^\top$ and $\vec{1} \cdot \vec{r} = \sum_{i=1}^{n_2} r_i = s$ where r_1, r_2, \dots, r_{n_2} are chosen at random from \mathbb{Z}_q , and s is the shared secret. $\mathbf{M}\vec{r} \in \mathbb{Z}_q^{n_1 \times 1}$ is the vector of n_1 shares of s and each one is labeled according to $\hat{\mathbf{M}}$.

The reconstruction of secret could be described as below: suppose $G^* \in \mathbb{A}$ is an authorization set and δ_{G^*} is the feature vector of G^* , we obtain $\vec{1} \in \text{span}(\mathbf{M}_\delta)$, which means there exists a constant set $\{\beta_u\}_{u \in n_1}$ such that

$$\sum_{i=1}^{n_1} \beta_i \mathbf{M}_i = \vec{1}, \quad (1)$$

and then $\sum_{i=1}^{n_1} \beta_i \mathbf{M}_i \cdot \vec{r} = \vec{1} \cdot \vec{r} = s$. Note that these constants $\{\beta_u\}_{u \in n_1}$ could be found in polynomial time.

4.3. Robust Fingerprinting Code

To address the problem of watermarking digital content, Boneh et al. introduced fingerprinting codes, which are a pair of the following two algorithms [34]:

- $FC.Gen(n, a)$: On input the number of users and security parameter a , output a tracing key α and a codebook $C \in \{0, 1\}^{n \times \ell}$ where each row c_i is the codeword for user i and ℓ is the fingerprinting code.
- $FC.Trace(tk, C^*)$: It requires tk and a collusion codeword C^* as input, outputs the identity set T of accused users \mathcal{U}_{acc} .

A fingerprinting code is t -collusion resistant means that there are t corrupted users \mathcal{U}_{cor} and their pirate codeword C^* generated by pooling their codewords together, such that:

$$\Pr[\mathcal{U}_{acc} = \emptyset \text{ or } \mathcal{U}_{cor} \not\subseteq \mathcal{U}_{acc} : \mathcal{U}_{acc} \leftarrow FC.Trace(tk, C^*)] \leq a$$

A fingerprinting code is δ -robust means that there exists at most $\delta\ell$ symbols '?' in codeword.

4.4. ABET

An ABET scheme consists of the following five algorithms:

- $ABET.Setup(n, 1^\lambda)$: It requires the security parameter λ and the number of participants n as input, outputs key pair (mpk, msk) and tracing key tk .
- $ABET.KGen(mpk, msk, S_i)$: It requires (mpk, msk) and the attribute set S_i of participant i as input, outputs the individual secret key sk_i .
- $ABET.Encrypt(mpk, \mathbb{A}, m)$: It requires mpk , access structure \mathbb{A} , and message m as input, outputs the ciphertext CT .
- $ABET.Decrypt(mpk, CT, sk_i)$: It requires mpk , CT , and sk_i as input, outputs m or \perp .
- $ABET.Trace^{Ds}(mpk, tk, S)$: It requires a pirate decoder D_S (associated with a set of attributes S), mpk and tk as input, outputs the identity set T of traitors.

5. Policy-Based Chameleon Hash with Traceability

In this section, we give the formal definition and security model of PCHT.

5.1. Formal Definition

We consider three entities in PCHT: an authority *auth*, an IoT data owner *DO*, and a transaction modifier *TM*. Specifically, it includes the following algorithms:

- $PCHT.Setup(n, 1^\lambda)$: It requires the security parameter λ and the number of participants n as input, outputs the key pair (pk, sk) .
- $PCHT.KGen(pk, sk, S_i)$: It requires (pk, sk) and attribute set $S_i \in \mathcal{U}$ as input, outputs the secret key sk_i .
- $PCHT.Hash(pk, m, \mathbb{A})$: It requires pk , message m , and \mathbb{A} as input, outputs randomness r , chameleon hash η , and ciphertext CT .
- $PCHT.Verify(pk, m, \eta, r)$: It requires pk , m , η , and r as input, outputs a bit $b \in \{0, 1\}$.
- $PCHT.Adapt(sk_{tm}, m', m, \eta, r, CT)$: It requires sk_{tm} , new message m' , m , η , r , and CT as input, outputs the new randomness r' , chameleon hash η' , and ciphertext CT' .
- $PCHT.Trace^{D_S}(pk, sk, S)$: It requires a pirate decoder D_S associated with a set of attributes S , (pk, sk) as input, outputs an index set $T \subseteq \{1, \dots, n\}$ of corrupted modifiers.

Correctness: The PCHT is correct if for all:

$$\begin{aligned}(pk, sk) &\leftarrow PCHT.Setup(n, 1^\lambda), \quad \lambda \in \mathbb{Z}, \\ sk_i &\leftarrow PCHT.KGen(pk, sk, S_i), \quad S_i \in \mathcal{U}, \\ (r, \eta, CT) &\leftarrow PCHT.Hash(pk, m, \mathbb{A}, sk_{do}), \quad m \in \mathcal{M}, \\ (r', \eta', CT') &\leftarrow PCHT.Adapt(sk_{tm}, m', m, r, \eta, CT),\end{aligned}$$

we have that:

$$PCHT.Verify(pk, m, \eta, r) = PCHT.Verify(pk, m', \eta', r') = 1.$$

5.2. Security Model

This section considers two security models of PCHT: indistinguishability and collision resistance.

Indistinguishability. Informally, the indistinguishability requires that the adversary can not be sure if the randomness of the chameleon hash was generated by the hashing or adapting algorithm. The security experiment is shown below.

$$\begin{aligned}\mathbf{Exp}_{PCHT, \mathcal{A}}^{IND}(1^\lambda) : \\ (sk, pk) &\leftarrow PCHT.Setup(n, 1^\lambda); \quad b \leftarrow \{0, 1\} \\ b' &\leftarrow \mathcal{A}^{\mathcal{O}_{HashOrAdapt}(sk, \cdot, \cdot, \cdot, b)}(pk) \\ &\text{return } 1, \text{ if } b' = b; \text{ else, return } 0\end{aligned}$$

$$\begin{aligned}\mathbf{Oracle } \mathcal{O}_{HashOrAdapt}(sk, S_i, m, m', \mathbb{A}, b) : \\ sk_i &\leftarrow PCHT.KGen(pk, sk, S_i) \\ (r_0, \eta_0, CT^0) &\leftarrow PCHT.Hash(pk, m', \mathbb{A}) \\ (r_1, \eta_1, CT^1) &\leftarrow PCHT.Hash(pk, m, \mathbb{A}) \\ r_1 &\leftarrow PCHT.Adapt(sk_i, m', m, \eta_1, r_1, CT^1) \\ &\text{return } (r_b, \eta_b, CT^b)\end{aligned}$$

If the following advantage for any PPT adversary \mathcal{A} is negligible, PCHT scheme is indistinguishability secure:

$$\mathbf{Adv}_{PCHT, \mathcal{A}}^{IND}(1^\lambda) = |\Pr[\mathbf{Exp}_{PCHT, \mathcal{A}}^{IND}(1^\lambda) = 1] - 1/2|.$$

Collision resistance. Informally, the collision resistance requires that the adversary could not find collisions for hashes computed with the access policies they are not satisfied with. The security experiment is shown below.

$\text{Exp}_{\text{PCHT}, \mathcal{A}}^{\text{CR}}(1^\lambda) :$
 $(sk, pk) \leftarrow \text{PCHT.Setup}(n, 1^\lambda); \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3 \leftarrow \emptyset$
 $(m^*, r^*, m^{*'}, r^{*'}, \eta^*) \leftarrow \mathcal{A}^{\mathcal{O}}(pk),$
 where $\mathcal{O} \leftarrow \{\mathcal{O}_{\text{KGen}}, \mathcal{O}_{\text{KGen}'}, \mathcal{O}_{\text{Hash}}, \mathcal{O}_{\text{Adapt}}\}.$
 return 1, if $1 = \text{PCHT.Verify}(pk, m^*, \eta^*, r^*)$
 $= \text{PCHT.Verify}(pk, m^{*'}, \eta^*, r^{*'}) \wedge (\eta^*, \cdot, \mathbb{A}) \in \mathcal{L}_3,$
 for some $\mathbb{A} \wedge m^* \neq m^{*'} \wedge \mathbb{A} \cap \mathcal{L}_1 = \emptyset \wedge (\eta^*, m^*, \cdot) \notin \mathcal{L}_3$
 else, return 0

Oracle $\mathcal{O}_{\text{KGen}}(pk, sk, S_i) :$
 $sk_i \leftarrow \text{PCHT.KGen}(pk, sk, S_i); \mathcal{L}_1 \leftarrow \mathcal{L}_1 \cup \{S_i\}$
 return sk_i
Oracle $\mathcal{O}_{\text{KGen}'}(pk, sk, S_i) :$
 $sk_i \leftarrow \text{PCHT.KGen}(pk, sk, S_i); \mathcal{L}_2 \cup \{i, sk_i\}$
 $i \leftarrow i + 1$
Oracle $\mathcal{O}_{\text{Hash}}(pk, m, \mathbb{A})$
 $(\eta, r, CT) \leftarrow \text{PCHT.Hash}(pk, m, \mathbb{A})$
 $\mathcal{L}_3 \leftarrow \mathcal{L}_3 \cup \{(\eta, m, \mathbb{A})\}$
 return (η, r, CT)
Oracle $\mathcal{O}_{\text{Adapt}}(m, m', \eta, r, j, CT, sk_{tm}) :$
 return \perp , if $(j, sk_{tm}) \notin \mathcal{L}_2$ for some sk_{tm}
 $r' \leftarrow \text{PCHT.Adapt}(pk, sk_{tm}, m, m', \eta, r, CT)$
 if $(\eta, m, \mathbb{A}) \in \mathcal{L}_3$ for some \mathbb{A} ,
 let $\mathcal{L}_3 \leftarrow \mathcal{L}_3 \cup \{(\eta, m', \mathbb{A})\}$
 return r'

If the following advantage is negligible, PCHT scheme is collision resistant:

$$\text{Adv}_{\text{PCHT}, \mathcal{A}}^{\text{CR}}(1^\lambda) = \Pr[\text{Exp}_{\text{PCHT}, \mathcal{A}}^{\text{CR}}(1^\lambda) = 1].$$

6. Security Analysis and Instantiation

6.1. Security Analysis

The construction of PCHT involves two cryptographic building blocks: chameleon hash with ephemeral trapdoor scheme with indistinguishability as well as collision resistance, and a traceable CP-ABE scheme with traceability. The security results of PCHT are shown below and the detailed proofs are also given.

Theorem 1. *If the underlying CHET is indistinguishable, PCHT scheme has indistinguishability secure.*

Proof. Let \mathcal{A} be a PPT adversary who can break the indistinguishability of PCHT with considerable advantage and \mathcal{B} be a PPT distinguisher that can break the indistinguishability of CHET with the following advantage: $\text{Adv}_{\text{PCHT}, \mathcal{A}}^{\text{IND}}(1^\lambda) = \text{Adv}_{\text{CHET}, \mathcal{B}}^{\text{IND}}(1^\lambda)$. Moreover, \mathcal{B} is given a HashOrAdapt oracle.

Setup. At the beginning, \mathcal{B} creates n participants for \mathcal{A} and honestly generates a secret key for each one.

Query. If \mathcal{A} queries (S_i, m, m', \mathbb{A}) , then \mathcal{B} obtains (r_b, η_b, CT^b) from his *HashOrAdapt* oracle. Finally, \mathcal{B} returns (r_b, η_b, CT^b) .

Guess. Both of \mathcal{A} and \mathcal{B} output a bit b . If \mathcal{A} guesses correctly, then \mathcal{B} can break the indistinguishability of CHET. \square

Theorem 2. *The PCHT scheme is collision-resistant if the traceable CP-ABE scheme is adaptive secure, and the underlying chameleon hash with ephemeral trapdoor is collision resistant.*

Proof. A sequence of games \mathbb{G}_i ($i = 0, \dots, 3$) are defined.

\mathbb{G}_0 : The original game for collision resistance.

\mathbb{G}_1 : Let q be an upper bound of the queries to Hash oracle. All queries are answered as in \mathbb{G}_0 , but the g -th one. In the g -th query, the encrypted message etd^* in CT^* is replaced by 0. First, two messages $M_0 = etd^*$, $M_1 = 0$ are submitted to the challenger by \mathcal{S} . Then, \mathcal{A} obtains the tuple (η^*, m^*, r^*, CT^*) from \mathcal{S} . If \mathcal{A} behaves significantly different in \mathbb{G}_0 and \mathbb{G}_1 , \mathcal{S} can use \mathcal{A} to break the semantic security of the traceable CP-ABE. Hence, we have

$$|\Pr[\mathcal{S}_0] - \Pr[\mathcal{S}_1]| \leq \text{Adv}_S^{\text{ABET}}(\lambda).$$

\mathbb{G}_2 : As \mathbb{G}_1 except that in the g -th query, if a valid collision is given by \mathcal{A} which was not previously returned by the Adapt oracle, \mathcal{S} outputs a random bit. First, all public parameters are honestly returned by \mathcal{S} . Then, \mathcal{S} obtains a randomness r from the Adapt oracle if \mathcal{A} submits an adapt query. If \mathcal{A} gives a collision $(\eta^*, m^*, r^*, CT^*, m^{*'}, r^{*'}, CT^{*'})$ in the g -th query and all the verifications hold, then a valid collision $(h^*, m^{*'}, r^{*'}, CT^{*'})$ is given to CHET by \mathcal{S} . Therefore, we have

$$|\Pr[\mathcal{S}_1] - \Pr[\mathcal{S}_2]| \leq \text{Adv}_S^{\text{CHET}}(\lambda).$$

Combining the above results, we have

$$\text{Adv}_A^{\text{PCHT}}(\lambda) \leq n(\lambda) \cdot (\text{Adv}_S^{\text{ABET}}(\lambda) + \text{Adv}_S^{\text{CHET}}(\lambda)).$$

\square

Traceability. The PCHT scheme is traceable since the underlying fingerprinting code is δ -robust. In the case that $\delta = 1$, D_S will decrypt correctly if all $\omega_i^{(j)} = 0$ or all $\omega_i^{(j)} = 1$ for $j \in \mathcal{U}_{cor}$. The complex case is that it can be either 1 or 0 for the position i . In summary, the traceability of fingerprinting code will return the indices set of traitors. For an imperfect decoder, it can also achieve traceability by the robustness of the fingerprinting codes. Since D_S has δ -correctness, it still works properly for many positions.

6.2. Instantiation

In this section, a practical instantiation of PCHT is given. First, we combine the efficient CP-ABE scheme [23] with Tardos code [35] (a robust fingerprinting code) to initiate a traceable CP-ABE scheme with traceability. Based on the asymmetric prime-order Type-III pairing, the chosen CP-ABE scheme in [23] is adaptively secure under the standard decision linear assumption and equips the characteristics such as unbounded ABE universes and constant decryption time. Then, the discrete logarithm-based CHET [22] is chosen to initiate CHET. The instantiation of the PCHT scheme is shown below.

- $\text{PCHT.Setup}(n, 1^\lambda)$: It takes a security parameter λ and the number of users n as input.
 - Consider a bilinear pairing: $\hat{e} : \mathbb{G} \times \mathbb{H} \rightarrow \mathbb{G}_T$, where g and h are the generators \mathbb{G} and \mathbb{H} , respectively.
 - Run $\text{CHET.KGen}(pp)$ to obtain the long-term secret/public key pair: (k, h^k) , where $k \in \mathbb{Z}_p^*$ and $pp \leftarrow \text{CHET.Setup}(1^\lambda)$.

- Let $\epsilon = 1/2^\lambda$ and run $FC.Gen(n, \epsilon)$ to obtain the tracing wordcode set codebook Φ and tracing key tk : $FC.Gen(n, \epsilon) \rightarrow \{tk, \Phi\}$, where $\Phi := \{\phi_1, \dots, \phi_n\}$.
- Pick $(a_1, a_2, b_1, b_2) \leftarrow_R \mathbb{Z}_p^*$, $(w_1, w_2, w_3) \leftarrow_R \mathbb{Z}_p$, compute: $A_1 := h^{a_1}$, $A_2 := h^{a_2}$, $H_1: \{0, 1\}^* \rightarrow \mathbb{G}$, $H_2: \{0, 1\}^* \rightarrow \mathbb{Z}_q$, $T_1 := \hat{e}\{g, h\}^{w_1 a_1 + w_3}$, $T_2 := \hat{e}\{g, h\}^{w_2 a_2 + w_3}$.
- Choose the dummy attributes $\{\text{Char}^u\}_{u \in \{0, 1\}}$, $\{\text{Char}_j\}_{j \in \{\ell\}}$ and return:
 $pk := \{h^k, g, h, A_1, A_2, T_1, T_2, H_1, H_2, \{\text{Char}^u\}_{u \in \{0, 1\}}, \{\text{Char}_j\}_{j \in \{\ell\}}\}$,
 $sk := \{k, a_1, a_2, b_1, b_2, g^{w_1}, g^{w_2}, g^{w_3}, \Phi, tk\}$.
- $PCHT.KGen(pk, sk, S_i)$: It takes pk , sk , and attribute set S_i of participant i as input:
 - Pick $(d_1, d_2) \in \mathbb{Z}_p^*$, $d = d_1 + d_2$, compute: $sk_0 := (h^{b_1 d_1}, h^{b_2 d_2}, h^d)$.
 - For $j \in \{1, \dots, \ell\}$ (ℓ is the code length), let $\phi_i^{(j)} \in \{0, 1\}$ be the j -th bit of ϕ_i and set $S_{i,j} = S_i \cup \text{Char}^{\phi_i^{(j)}} \cup \text{Char}_j$, for all $s \in S_{i,j}$ and $t = \{1, 2\}$, compute:

$$sk_{s,t} := H_1(s1t)^{b_1 d_1 / a_t} \cdot H_1(s2t)^{b_2 d_2 / a_t} \cdot H_1(s3t)^{d / a_t} \cdot g^{v_s / a_t},$$
 where $v_s \in \mathbb{Z}_q$. Set $sk_s := \{sk_{s,1}, sk_{s,2}, g^{-\sigma_s}\}$.
 - Pick $v' \in \mathbb{Z}_q$, for $t = \{1, 2\}$, compute:
 $sk'_t := g^{w_t} \cdot H_1(011t)^{b_1 d_1 / a_t} \cdot H_1(012t)^{b_2 d_2 / a_t} \cdot H_1(013t)^{d / a_t} \cdot g^{v' / a_t}$.
 - Then set: $sk' := (sk'_1, sk'_2, g^{w_3} \cdot g^{-v'})$. Finally, the decryption key for $S_{i,j}$ is:
 $sk_{i,j} := (sk_0, \{sk_s\}_{s \in S_{i,j}}, sk')$.
 - Return the secret key sk_i of participant i : $sk_i := (\{k\}, \{sk_{i,j}\}_{j \in \{\ell\}})$.
- $PCHT.Hash(pk, m, \mathbb{A}, sk_{do})$: It takes the master public key pk , message m , access structure \mathbb{A} , secret key of IoT data owner sk_{do} as input:
 - Pick a randomness $r \in \mathbb{Z}_p^*$ and a short bit-string etd as ephemeral trapdoor to obtain a chameleon hash η : $h' = h^{H_2(etd)}$, $\eta = h^{k \cdot r} \cdot h'^m$.
 - Pick $\tau_1, \tau_2 \leftarrow \mathbb{Z}_p$ and $\tau = \tau_1 + \tau_2$, compute: $ct_0 := (A_1^{\tau_1}, A_2^{\tau_2}, h^\tau)$.
 - Choose a random position $pos \in \{\ell\}$ and set $\bar{\mathbb{A}}_u = \mathbb{A} \wedge \{\text{Char}^u\} \wedge \{\text{Char}_{pos}\}$, where $u \in \{0, 1\}$.
 - Let $z = 1, 2, 3$ and $u = 0, 1$, suppose $\bar{\mathbb{A}}_u$ has n_1 rows and n_2 columns, for $\forall v \in \{1, \dots, n_1\}$, compute:
 $ct_{u,v,z} := H_1(\pi(v)z1)^{x_1} \cdot H_1(\pi(v)z2)^{x_2} \cdot \prod_{j=1}^{n_2} [H_1(0jz1)^{x_1} \cdot H_1(0jz2)^{x_2}]^{\bar{\mathbb{A}}_{u,v,j}}$,
 where $\bar{\mathbb{A}}_{u,v,j}$ denotes the (v, j) th element of $\bar{\mathbb{A}}_u$. Set $ct_{u,v} := (ct_{u,v,1}, ct_{u,v,2}, ct_{u,v,3})$.
 - Generate a ciphertext on message $msg := \{r, etd\}$ and compute: $ct' := T_1^{\tau_1} \cdot T_2^{\tau_2} \cdot msg$.
 - Finally, it outputs $CT_u = (ct_0, \{ct_{u,v}\}_{v \in n_1}, ct')$. Return $(pos, CT_0, CT_1, r, h', \eta)$.
- $PCHT.Verify(m, \eta, r, h')$: It takes message m , chameleon hash η , r , h' as input. Return 1 if $\eta = h^{k \cdot r} \cdot h'^m$; otherwise, return 0.
- $PCHT.Adapt(sk_{tm}, m', m, pos, CT_0, CT_1, r, h', \eta)$: It takes the secret key of modifier sk_{tm} , new message m' , message m , ciphertext CT , randomness r , h' , chameleon hash η as input:
 - Check 1 $\stackrel{?}{=} PCHT.Verify(m, \eta, r, h')$.
 - If $\phi_i^{pos} = 0$, then pick CT_0 to perform the following using decryption key $sk_{tm,pos}$; otherwise (i.e., $\phi_i^{pos} = 1$), the same for CT_1 .
 - Recall that the set of attributes in $sk_{tm,pos}$ satisfies the MSP $(\bar{\mathbb{A}}_0, \rho)$, then there exist constants $\{\gamma_i\}_{i \in I}$ that satisfies the Equation (1) in Section 1, compute:
 $\mathcal{A} := ct' \cdot \hat{e}(\prod_{i \in I} ct_{i,1}^{\gamma_i}, sk_{0,1}) \cdot \hat{e}(\prod_{i \in I} ct_{i,2}^{\gamma_i}, sk_{0,2}) \cdot \hat{e}(\prod_{i \in I} ct_{i,3}^{\gamma_i}, sk_{0,3})$,
 $\mathcal{B} := \hat{e}(sk'_1 \cdot \prod_{i \in I} sk_{\pi(i),1}, ct_{0,1}) \cdot \hat{e}(sk'_2 \cdot \prod_{i \in I} sk_{\pi(i),2}, ct_{0,2}) \cdot \hat{e}(sk'_3 \cdot \prod_{i \in I} sk_{\pi(i),3}, ct_{0,3})$, where $sk_{0,1}, sk_{0,2}, sk_{0,3}$ denote the first, second, and third element of sk_0 ; the same rule applies to ct_0 .
 - Derive the ephemeral trapdoor etd and randomness r from \mathcal{A}/\mathcal{B} , and then compute: $r' := r + (m - m') \cdot H_2(etd)/k$.

- Run $PCHT.Hash(pk, m', \mathbb{A}, sk_{tm})$ to obtain position pos' , ciphertext CT'_0 , and CT'_1 on $msg' := (r', etd)$ and return $(pos', CT'_0, CT'_1, r', h', \eta)$.
- $PCHT.Trace^{Ds}(pk, sk, S)$: Suppose the trace algorithm has black-box access to pirate decoder D_S . The authority takes (pk, sk) as input:
 - To obtain the identity set T of traitors, the *auth* performs as follows for each j in $\{\ell\}$:
 - * Choose an access policy \mathbb{B} which is only satisfied by S and not satisfied by any subset of S ;
 - * Set $\mathbb{B}_u := \mathbb{B} \wedge \{Attr_j\} \wedge \{Attr^u\}$, where $u \in \{0, 1\}$;
 - * Let $N = O(\lambda^2 \ln \ell)$ and repeat the following steps for N times: pick two random message m, m' , compute:

$$\begin{aligned}
 CT_0 &\leftarrow PCHT.Hash(pk, m, \mathbb{B}_0), \\
 CT_1 &\leftarrow PCHT.Hash(pk, 0, \mathbb{B}_1), \\
 CT'_0 &\leftarrow PCHT.Hash(pk, m', \mathbb{B}_0), \\
 CT'_1 &\leftarrow PCHT.Hash(pk, m', \mathbb{B}_1).
 \end{aligned}$$

Set $CT := (j, CT_0, CT_1)$ and $CT' := (j, CT'_0, CT'_1)$. If $D_S(CT) = m$, set $\phi^j = 0$; else if $D_S(CT') = m'$ for more than $\sqrt{\lambda}$ times, set $\phi^j = 1$; else, set $\phi^j = '?'$.

- * Set the unauthorized codeword $\phi_* = \{\phi^1 \dots \phi^\ell\}$, and run $FC.Trace(tk, \phi_*)$ to obtain the identity set T of traitors:

$$T \leftarrow FC.Trace(tk, \phi_*).$$

7. PCHT-Based Blockchain Rewriting Scheme for IoT

In this section, we first introduce the application of PCHT for blockchain rewriting and then present a PCHT-based blockchain rewriting scheme for IoT.

7.1. Application for Blockchain Rewriting

We give an application of PCHT for blockchain rewriting, which makes the blockchain remain intact even if certain mutable transactions have been modified. As Figure 2 shows, there are four transactions in the block B_i and their hash values are the leaf nodes of a Merkle tree rooted at $txroot$. Among them, $T_{(i,1)}$ and $T_{(i,3)}$ are mutable transactions associated with different access structures, $T_{(i,2)}$ and $T_{(i,4)}$ are immutable transactions. When the mutable transaction needs to be modified, a transaction modifier could compute valid randomness r without changing its hash value and $txroot$ as long as the attributes S_i embedded in his PCHT-based decryption key sk_{dm} are accepted by the access structure \mathbb{A} of this mutable transaction. After modifying, the transaction modifier broadcasts this transaction and randomness r to the network. Each participant could validate the correctness of this mutable transaction and then update its local ledger of the blockchain with new content and randomness.

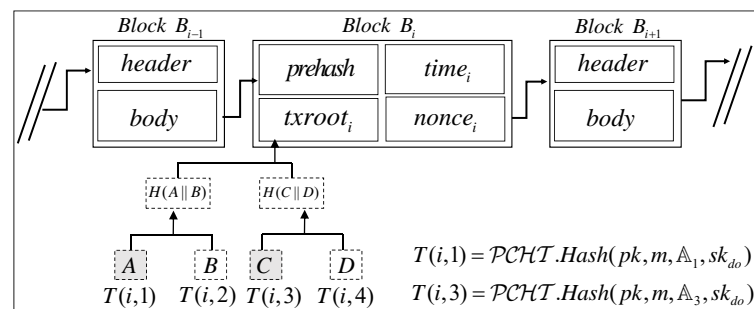


Figure 2. PCHT-based blockchain rewriting.

7.2. System Model

As shown in Figure 1, we consider three types of entities for PCHT-based blockchain rewriting: the authority *auth*, the IoT data owner *DO*, and the transaction modifier *TM*. To clearly describe the PCHT-based blockchain rewriting scheme, we introduce each step in detail as follows.

- System initialization. There are two phases in system initialization:
 - Master key pair generation: the *auth* obtains (pk, sk) by running $PCHT.Setup(n, 1^\lambda)$, and only publishes pk to participants in the IoT blockchain.
 - Member key generation: the *auth* generates individual secret key sk_i for each participant with their own attributes set S_i by running $PCHT.KGen(pk, sk, S_i)$.
- Mutable transaction publication. It consists of the two following phases:
 - Mutable transaction generation: *DO* generates a mutable transaction that includes the message m , chameleon hash η , randomness r , and ciphertext CT by running $PCHT.Hash(pk, m, \mathbb{A})$. Then, *DO* broadcasts this transaction to other participants in the blockchain for IoT.
 - Mutable transaction verification: each participant could validate the transaction by running the verifying algorithm. If the verification algorithm returns 1, this participant could append it to its local ledger and broadcast it continually.
- Mutable transaction rewriting. There are two following phases:
 - Mutable transaction rewriting: To rewrite the transaction content from m to m' , the transaction modifier *TM* whose attributes are accepted by the access structure \mathbb{A} could run $PCHT.Adapt(sk_{tm}, m, m', CT, \eta, r)$ to compute a valid hash collision and rewrite the transaction successfully. Then, the *TM* broadcasts the modified transaction publicly.
 - Mutable transaction verification: Each participant could validate the new transaction by running $PCHT.Verify(m', \eta, r')$. He would update his local copy with the message m' if the transaction is valid and broadcast it to other participants continually.
- Traitor tracing. It could be completed by the following step:
 - Corrupted modifiers tracing: The *auth* could obtain the identity set T of corrupted modifiers who produce the pirate decoder by running $PCHT.Trace^{Ds}(pk, sk, S)$.

7.3. Threat Model

In this context, we assume the authority *auth* would honestly generate the secret key sk_i for participant i and perform his traceability duties honestly during traitor tracing. Moreover, the IoT data owner *DO* would honestly publish the mutable transaction.

Our main security goals are to guarantee indistinguishability of randomness r and chameleon hash η , protect against adversaries who try to update m without the long-term and/or ephemeral trapdoor and ensure the traceability of *auth*. The corrupted modifiers may launch the following various types of attacks. In this paper, we do not consider network attacks in the permissioned blockchain (e.g., replay attacks, denial of service, etc.), existing works have already focused on them [43].

Indistinguishability. In reality, the adversary could identify chameleon hash has been modified or not to learn the additional knowledge. To avoid this, this property has been formalized by introducing the security model of *indistinguishability* and a chameleon hash with ephemeral trapdoor scheme with indistinguishability is used to construct the PCHT.

Collision resistance. This property ensures the security of the mutable transaction. The adversary may want to rewrite the mutable transaction when they are not satisfied with the access policies embedded in the transaction. Therefore, we formalize this property by introducing the security model of *collision resistance*. Moreover, a traceable CP-ABE scheme with adaptive security and a collision-resistant CHET scheme are chosen to construct the PCHT.

Traceability. The corrupted modifiers may collude to produce the pirate decoder by abusing their decryption keys. To solve this, we use a traceable CP-ABE scheme with traceability as well as δ -robust fingerprinting code to construct PCHT.

8. Implementation and Evaluation

In this section, we first implement the proposed instantiation of PCHT using Charm framework [44] and evaluate its performance on a personal computer with Ubuntu 18.04, Intel Core i7-8700@3.20 GHz, and 4 GB RAM. Based on hyperledger fabric v1.4, the PCHT-based redactable blockchain platform has also been implemented using java-1.8.0, go-1.15.6, docker-20.10, docker-compose-2.3.4, vue-2.9.6, including one CA peer, one ordering peer, and seven committing peers.

In the implementation of PCHT, we use the MNT224 curve for pairing, which is the Type-III curve in PBC and offers a 96-bit security level [23]. The experimental performance of key generation, hashing, and adaption algorithms compared with the PCHBA scheme in [24] is present in Figure 3. Overall, the runtime of these algorithms is proportional to the number of attributes or the size of policies. Since we set the code length $\ell = 5$ in the test, the runtime of the key generation algorithm in PCHT is about five times as long as it in PCHBA, which is because the *auth* needs to generate the decryption key for each code and the runtime of key generation algorithm would increase with the length of code length. In our opinion, the key generation phase only takes place during system initialization and it is critical to the traceability of PCHT. Hence, the reasonable time cost is acceptable. For the hashing and adaption algorithms, the number of attributes ranges from 10 to 100 and the runtime of PCHT is about twice that in PCHBA, which is because there are two ciphertexts that need to be generated and the time cost is moderate. Given the restricted computing power of certain IoT devices, IoT data sharing is collected by data owners while transaction rewriting operations are carried out by full nodes on the IoT blockchain, which possess adequate computational resources for executing the hashing and adaptation algorithms. The rare occurrence of transaction rewriting in the IoT system means that it is unlikely to have any significant detrimental effect on the system's performance.

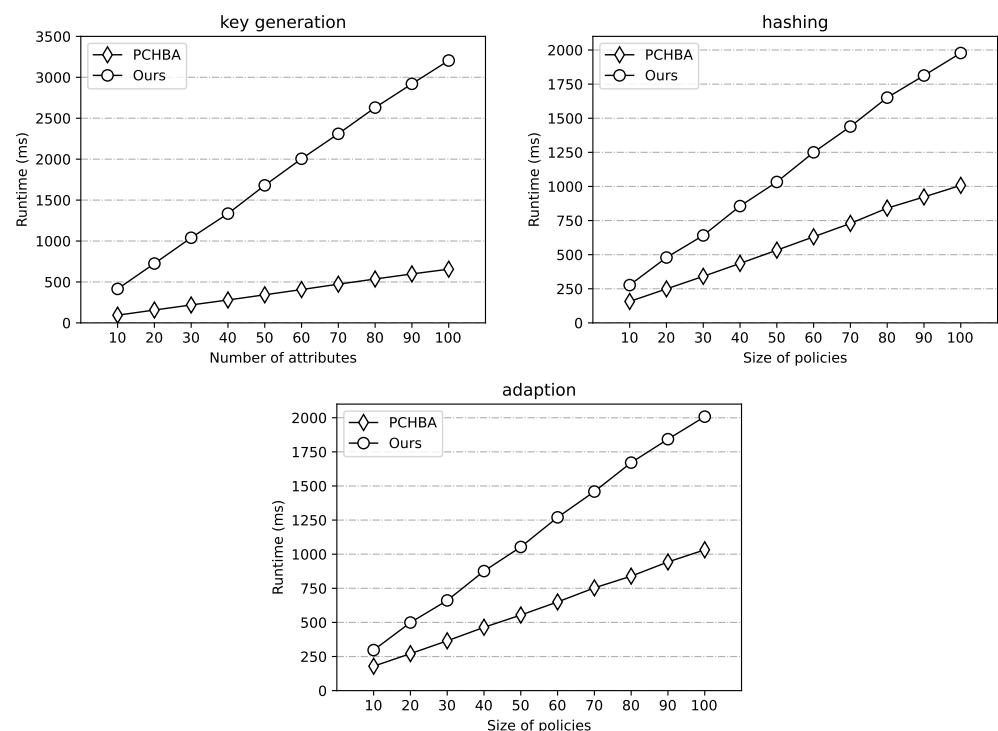


Figure 3. Experimental performance.

The PCHT-based redactable blockchain platform has also been deployed in this paper. To evaluate its performance, we use Tape as the throughput testing tool and the number of transactions included in each block is set to 160. Moreover, the runtime of hashing and adaption algorithms in this test is about 200 ms. Set the proportion of mutable transaction volume to the total transaction volume is 5%, 10%, 15%, 20%, respectively, and we obtain the following test results. As shown in Figure 4, the transaction throughput decreases as the mutable transaction ratio increases, which means the runtime of PCHT imposes a non-negligible impact on system performance. Considering the fact that the mutable transactions happen infrequently in the IoT blockchain, the lower transaction throughput compared with a single application blockchain is reasonable and acceptable.

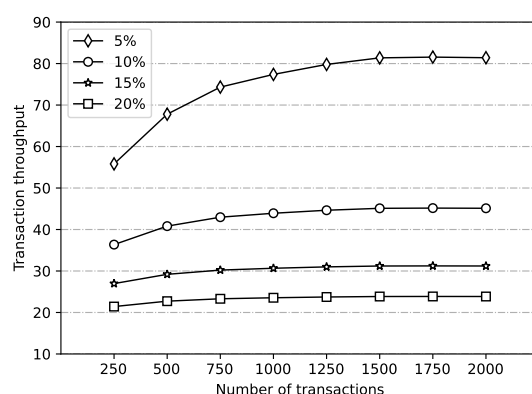


Figure 4. The transaction throughput under different transaction volume.

9. Conclusions

The Internet of Things (IoT) enables the collection of data from multiple parties through sensors, and blockchain technology can serve as an ideal solution for establishing trust in a multi-center cooperation framework. In order to ensure the redaction of data and traceability of any malicious transaction tampering, we propose a policy-based chameleon hash with traceability, known as PCHT, and introduce a PCHT-based blockchain rewriting scheme. Our experimental analysis demonstrates that this scheme does not impose a computational burden on IoT devices. Future work could focus on decentralizing the authority and extending our generic construction to support authority accountability.

Author Contributions: Conceptualization, P.D.; formal analysis, P.D.; funding acquisition, Z.M.; methodology, P.D.; project administration, Z.M.; software, P.D. and J.W.; supervision, Y.Z., Z.M. and S.L.; validation, P.D., J.W. and Y.Z.; writing—original draft, P.D.; writing—review and editing, J.W., Y.Z., Z.M. and S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Plan in China of funder grant number 2020YFB1005500 and Beijing Natural Science Foundation of funder grant number M21034. The APC was funded by the National Key Research and Development Plan in China of funder grant number 2020YFB1005500.

Data Availability Statement: Data sharing not applicable.

Acknowledgments: The research work is supported by the National Key Research and Development Plan in China (Grant No. 2020YFB1005500) and Beijing Natural Science Foundation (Grant No. M21034).

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

References

1. Singh, S.; Rathore, S.; Alfarraj, O.; Tolba, A.; Yoon, B. A framework for privacy-preservation of IoT healthcare data using Federated Learning and blockchain technology. *Future Gener. Comput. Syst.* **2022**, *129*, 380–388. [\[CrossRef\]](#)
2. Mall, P.; Amin, R.; Das, A.K.; Leung, M.T.; Choo, K.K.R. PUF-based authentication and key agreement protocols for IoT, WSNs, and Smart Grids: A comprehensive survey. *IEEE Internet Things J.* **2022**, *9*, 8205–8228. [\[CrossRef\]](#)

3. Laghari, A.A.; Wu, K.; Laghari, R.A.; Ali, M.; Khan, A.A. A review and state of art of Internet of Things (IoT). *Arch. Comput. Methods Eng.* **2022**, *29*, 1395–1413. [\[CrossRef\]](#)
4. Laghari, A.A.; Khan, A.A.; Alkanhel, R.; Elmannai, H.; Bourouis, S. Lightweight-BIoV: Blockchain Distributed Ledger Technology (BDLT) for Internet of Vehicles (IoVs). *Electronics* **2023**, *12*, 677. [\[CrossRef\]](#)
5. Waqas, M.; Kumar, K.; Laghari, A.A.; Saeed, U.; Rind, M.M.; Shaikh, A.A.; Hussain, F.; Rai, A.; Qazi, A.Q. Botnet attack detection in Internet of Things devices over cloud environment via machine learning. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6662. [\[CrossRef\]](#)
6. Ahanger, T.A.; Aljumah, A.; Atiquzzaman, M. State-of-the-art survey of artificial intelligent techniques for IoT security. *Comput. Netw.* **2022**, *206*, 108771. [\[CrossRef\]](#)
7. Rehman Javed, A.; Jalil, Z.; Atif Moqurrah, S.; Abbas, S.; Liu, X. Ensemble adaboost classifier for accurate and fast detection of botnet attacks in connected vehicles. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*, e4088. [\[CrossRef\]](#)
8. Li, R.; Song, T.; Mei, B.; Li, H.; Cheng, X.; Sun, L. Blockchain for large-scale internet of things data storage and protection. *IEEE Trans. Serv. Comput.* **2018**, *12*, 762–771. [\[CrossRef\]](#)
9. Wang, C.; Cai, Z.; Li, Y. Sustainable blockchain-based digital twin management architecture for IoT devices. *IEEE Internet Things J.* **2022**. [\[CrossRef\]](#)
10. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Cryptography Mailing List. 2008. Available online: <https://metzdowd.com> (accessed on 31 October 2008).
11. Cao, B.; Zhang, Z.; Feng, D.; Zhang, S.; Zhang, L.; Peng, M.; Li, Y. Performance analysis and comparison of PoW, PoS and DAG based blockchains. *Digit. Commun. Netw.* **2020**, *6*, 480–485. [\[CrossRef\]](#)
12. Liu, Y.; Zhang, C.; Yan, Y.; Zhou, X.; Tian, Z.; Zhang, J. A semi-centralized trust management model based on blockchain for data exchange in iot system. *IEEE Trans. Serv. Comput.* **2022**. [\[CrossRef\]](#)
13. Zhang, G.; Zhang, X.; Bilal, M.; Dou, W.; Xu, X.; Rodrigues, J.J. Identifying fraud in medical insurance based on blockchain and deep learning. *Future Gener. Comput. Syst.* **2022**, *130*, 140–154. [\[CrossRef\]](#)
14. Elhence, A.; Goyal, A.; Chamola, V.; Sikdar, B. A Blockchain and ML-Based Framework for Fast and Cost-Effective Health Insurance Industry Operations. *IEEE Trans. Comput. Soc. Syst.* **2022**. [\[CrossRef\]](#)
15. Ma, Z.; Jiang, M.; Gao, H.; Wang, Z. Blockchain for digital rights management. *Future Gener. Comput. Syst.* **2018**, *89*, 746–764. [\[CrossRef\]](#)
16. Florea, A.I.; Anghel, I.; Cioara, T. A Review of Blockchain Technology Applications in Ambient Assisted Living. *Future Internet* **2022**, *14*, 150. [\[CrossRef\]](#)
17. Wei, X.; Yan, Y.; Guo, S.; Qiu, X.; Qi, F. Secure Data Sharing: Blockchain-Enabled Data Access Control Framework for IoT. *IEEE Internet Things J.* **2022**, *9*, 8143–8153. [\[CrossRef\]](#)
18. Weerapanisit, P.; Trilles, S.; Huerta, J.; Painho, M. A Decentralized Location-Based Reputation Management System in the IoT Using Blockchain. *IEEE Internet Things J.* **2022**, *9*, 15100–15115. [\[CrossRef\]](#)
19. Qiu, J.; Tian, Z.; Du, C.; Zuo, Q.; Su, S.; Fang, B. A survey on access control in the age of internet of things. *IEEE Internet Things J.* **2020**, *7*, 4682–4696. [\[CrossRef\]](#)
20. Voigt, P.; Von dem Bussche, A. The EU general data protection regulation (GDPR). In *A Practical Guide*, 1st ed.; Springer International Publishing: Cham, Switzerland, 2017; Volume 10, pp. 10–5555.
21. Ateniese, G.; Magri, B.; Venturi, D.; Andrade, E. Redactable Blockchain—or—Rewriting History in Bitcoin and Friends. In Proceedings of the IEEE European Symposium on Security and Privacy, Paris, France, 26–28 April 2017; pp. 111–126.
22. Camenisch, J.; Derler, D.; Krenn, S.; Pöhls, H.C.; Samelin, K.; Slamanig, D. Chameleon-Hashes with Ephemeral Trapdoors. In Proceedings of the Public-Key Cryptography, Amsterdam, The Netherlands, 28–31 March 2017; pp. 152–182.
23. Agrawal, S.; Chase, M. FAME: Fast Attribute-Based Message Encryption. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; Association for Computing Machinery: New York, NY, USA, 2017.
24. Tian, Y.; Li, N.; Li, Y.; Szalachowski, P.; Zhou, J. Policy-based chameleon hash for blockchain rewriting with black-box accountability. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 7–11 December 2020; pp. 813–828.
25. Panwar, G.; Vishwanathan, R.; Misra, S. ReTRACe: Revocable and traceable blockchain rewrites using attribute-based cryptosystems. In Proceedings of the 26th ACM Symposium on Access Control Models and Technologies, Virtual, 16–18 June 2021; pp. 103–114.
26. Boneh, D.; Kiayias, A.; Montgomery, H.W. Robust fingerprinting codes: A near optimal construction. In Proceedings of the Tenth Annual ACM Workshop on Digital Rights Management, Chicago, IL, USA, 4 October 2010; pp. 3–12.
27. Lai, J.; Tang, Q. Making any attribute-based encryption accountable, efficiently. In Proceedings of the European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, 3–7 September 2018; Springer: Cham, Switzerland, 2018; pp. 527–547.
28. Boneh, D.; Waters, B. A fully collusion resistant broadcast, trace, and revoke system. In Proceedings of the 13th ACM conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006; pp. 211–220.

29. Lewko, A.; Waters, B. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In Proceedings of the 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 180–198.
30. Seo, J.H.; Cheon, J.H. Fully secure anonymous hierarchical identity-based encryption with constant size ciphertexts. *Cryptol. ePrint Arch.* **2011**, 21. Available online: <https://eprint.iacr.org/2011/021> (accessed on 17 February 2023).
31. Ning, J.; Dong, X.; Cao, Z.; Wei, L.; Lin, X. White-box traceable ciphertext-policy attribute-based encryption supporting flexible attributes. *IEEE Trans. Inf. Forensics Secur.* **2015**, 10, 1274–1288. [[CrossRef](#)]
32. Liu, Z.; Cao, Z.; Wong, D.S. Blackbox traceable CP-ABE: How to catch people leaking their keys by selling decryption devices on eBay. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; pp. 475–486.
33. Boneh, D.; Naor, M. Traitor tracing with constant size ciphertext. In Proceedings of the 15th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 27–31 October 2008; pp. 501–510.
34. Boneh, D.; Shaw, J. Collusion-secure fingerprinting for digital data. *IEEE Trans. Inf. Theory* **1998**, 44, 1897–1905. [[CrossRef](#)]
35. Tardos, G. Optimal probabilistic fingerprint codes. *J. ACM* **2008**, 55, 1–24. [[CrossRef](#)]
36. Wu, C.; Ke, L.; Du, Y. Quantum resistant key-exposure free chameleon hash and applications in redactable blockchain. *Inf. Sci.* **2021**, 548, 438–449. [[CrossRef](#)]
37. Puddu, I.; Dmitrienko, A.; Capkun, S. μ chain: How to Forget without Hard Forks. *Cryptol. ePrint Arch.* **2017**, 2017, 106.
38. Krawczyk, H.; Rabin, T. Chameleon hashing and signatures. *IACR Cryptol. ePrint Arch.* **1998**, 1998, 10.
39. Derler, D.; Samelin, K.; Slamanig, D.; Striecks, C. Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. *NDSS* **2019**. [[CrossRef](#)]
40. Zhang, Z.; Li, T.; Wang, Z.; Liu, J. Redactable transactions in consortium blockchain: Controlled by multi-authority CP-ABE. In Proceedings of the Information Security and Privacy: 26th Australasian Conference, Virtual Event, 1–3 December 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 408–429.
41. Ma, J.; Xu, S.; Ning, J.; Huang, X.; Deng, R.H. Redactable blockchain in decentralized setting. *IEEE Trans. Inf. Forensics Secur.* **2022**, 17, 1227–1242. [[CrossRef](#)]
42. Chase, M. Multi-authority attribute based encryption. In Proceedings of the Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, 21–24 February 2007; Springer: Berlin/Heidelberg, Germany, 2007; pp. 515–534.
43. Altarawneh, A.; Sun, F.; Brooks, R.R.; Hambolu, O.; Yu, L.; Skjellum, A. Availability analysis of a permissioned blockchain with a lightweight consensus protocol. *Comput. Secur.* **2021**, 102, 102098. [[CrossRef](#)]
44. Akinyele, J.A.; Garman, C.; Miers, I.; Pagano, M.W.; Rushanan, M.; Green, M.; Rubin, A.D. Charm: A framework for rapidly prototyping cryptosystems. *J. Cryptogr. Eng.* **2013**, 3, 111–128. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.