Received: 27 September 2024 | Revised: 5 November 2024 | Accepted: 4 December 2024 | Published online: 24 December 2024

Journal of Data Science and Intelligent Systems
2024, Vol. 00(00) 1–9
DOI: 10.47852/bonviewJDSIS42024450

BON VIEW PUBLISHING

RESEARCH ARTICLE

A Dynamic Trapdoor Redactable Blockchain Schemes

Shuqiao Li[1], Xiaoming Hu[1],*, Yan Liu[1] and Shuangjie Bai[1]

[1]*School of Computer and Information Engineering, Shanghai Polytechnic University, China*

Abstract: The permanent storage of blockchain data exposes certain security risks in practical applications, necessitating the ability to edit data in specific scenarios. The chameleon hash function, as a trapdoor one-way hash function, has emerged as a key technology for achieving redactable blockchains. However, existing redactable blockchain schemes based on chameleon hash function face security vulnerabilities related to trapdoor misuse and leakage. This paper proposes a dynamic trapdoor redactable blockchain schemes, utilizing a dynamic trapdoor chameleon hash (DTCH) that allows for the dynamic update of trapdoors while overcoming the inherent immutability of blockchain, thus reducing the risk of malicious nodes retaining the trapdoor long-term. The proposed scheme employs controllable randomness, where the trapdoor key automatically updates after a certain period or upon use. In this scenario, authorized nodes cannot successfully find hash collisions with old keys, effectively revoking their authorization. Consequently, only users with valid trapdoor keys can modify the data on the chain. Relevant security tests and experimental results indicate that the proposed scheme offers advantages in security and maintains good efficiency at various stages, achieving higher security guarantees with minimal computational cost.

Keywords: data management, redactable blockchain, chameleon hash function, dynamic trapdoor

1. Introduction

In 2008, Nakamoto first proposed Bitcoin, an electronic payment system based on Peer-to-Peer (P2P) network and consensus mechanism [1]. As the core technology of Bitcoin, blockchain has attracted widespread attention from academia, industry, and government departments due to its characteristics of decentralization, transparency, and immutability [2]. Among them, immutability means that once data are recorded on the blockchain, it cannot be deleted or modified, providing a guarantee for the reliability and credibility of blockchain data [3]. However, this permanent storage feature also brings many security risks. Malicious users may upload sensitive or illegal data to the blockchain, seriously affecting the healthy development of the network. In addition, data protection laws and regulations promulgated by government agencies require the editable nature of data to promote the rapid development and widespread application of blockchain technology [4]. Therefore, researching redactable schemes for blockchain data holds significant theoretical importance and practical application value.

In order to solve the contradiction between the immutability of data on the blockchain and data supervision, scholars have proposed the concept of redactable blockchain, which has been continuously researched and developed in recent years. Lou et al. conducted a systematic investigation and analysis of current redactable blockchain schemes and divided redactable blockchain technologies into chameleon hash-based schemes and non-chameleon hash-based schemes, where the latter includes consensus voting, mutable transactions, and other consensus algorithms [5]. The redactable blockchain constructed by chameleon hash function offers advantages such as low cost and high efficiency, making it the predominant method for constructing redactable blockchains [6]. This is primarily attributed to the unique property of chameleon hash, which possesses a trapdoor that allows the input of the hash function to be altered without changing the hash value. For non-trapdoor holders, chameleon hash maintains collision resistance similar to that of standard hash. In redactable blockchain system, the holder of chameleon hash trapdoor has the edit permission to modify data [7].

Although chameleon hash function provides technical support for redactable blockchain, there are still many problems to be solved. One of the first issues is how to achieve data trust in redactable blockchains. The advantage of blockchain is data tamper-proof, and editability seems to break this advantage. If the data on the chain can be arbitrarily modified by anyone at any time, the authenticity and reliability of the data cannot be guaranteed, and data trust is out of the question. Therefore, redactable blockchains that support data modification need to establish secure and perfect trust and supervision mechanisms to prevent the abuse of editing permissions [8].

To address the above challenges, the main contributions of this paper are as follows:

1) We propose a chameleon hash function supporting dynamic trapdoor. The trapdoor abolition and update algorithm are added to the chameleon hash function, which can realize the dynamic update of trapdoor when breaking through the inherent immutability of blockchain. Even if the key is stolen, the key held by the thief will become invalid after a period of time, thereby reducing the risk of malicious nodes holding

*Corresponding author: Xiaoming Hu, School of Computer and Information Engineering, Shanghai Polytechnic University, China. Email: xmhu@sspu.edu.cn

trapdoor for a long time. It solves the security risks of trapdoor abuse and trapdoor leakage and has higher security.
2) We design the identity-based user key generation algorithm. The algorithm generates the corresponding user key after identifying the user's identity and sets the validity period. The authorization is effectively revoked with controllable randomness, so that only the user with the legal trapdoor key can change the data that has been linked, which has the effect of dynamic rights management.
3) We prove the security of the proposed scheme and evaluate the performance of DTCH. The results show that our scheme is security and effectiveness. Compared with other schemes, our scheme achieves higher security guarantee with lower computation cost.

The rest of this paper is organized as follows. In Section 2, we elaborate literature review; In Section 3, we review the preliminary knowledge used throughout the paper. In Section 4, we present the system model and the concrete construction of DTCH. In Section 5, we describe the security analysis of DTCH. In Section 6, we provide the performance analysis of DTCH. In Section 7, we present discussion of this paper. Section 8 concludes the paper.

2. Literature Review

In 2017, Ashritha et al. [9] proposed a blockchain historical data editing method based on the chameleon hash algorithm. The chameleon hash function can ensure the consistency of hash values through pre left trapdoors, allowing users with chameleon hash algorithm trapdoors to edit the data of historical blocks. However, there is a problem of centralized holding of trapdoors.

In 2019, Derler et al. [10] proposed a scheme based on temporary trapdoors (CHET) and attribute-based encryption (ABE), in which the authors proposed a new cryptographic primitive called policy-based chameleon hash (PCH). In PCH, only the modifier who knows the long-term trapdoor and temporary trapdoor can modify the transaction, but the cost is high. Huang et al. [11] proposed threshold chameleon hash (TCH), which allows a group of authorized sensors to write and rewrite the blockchain without causing any hard forks, but poses significant security risks.

In 2020, Huang et al. [12] proposed the first revocable chameleon hash (RCH), which enables a transient trapdoor to detect collisions without any cooperation, periodically expires for submitted hashes, and a transient trapdoor to prevent any abuse of editorial power, but there is a trapdoor exposure issue.

In 2021, Jia et al. [13] introduced a stateful chameleon hash function in their article, which divides the key into a master key and multiple slave keys. The master key is used for regulatory agencies, and the slave key is used for users. Each user is assigned a corresponding state by the regulatory agency while being assigned a slave key. When a user initiates a modification request, the verification node not only verifies the legitimacy of the request but also verifies whether the user's key state is available. This scheme can effectively prevent malicious nodes from doing evil and improve system security, but it still relies on trusted regulatory agencies. Once the regulatory agency is dishonest, it will cause huge damage to the system. Xu et al. [14] introduced the concept of revocable PCH (RPCH) based on Derler, which revokes the ability of certain users to decrypt symmetric encryption keys by changing the access policy. However, their scheme does not truly achieve revocability, only revoking the ability of users who obtain trapdoors for decryption. Panwar et al. [15] proposed an efficient framework called ReTRACe for transaction-level blockchain rewriting, designed by combining a new revocable chameleon hash with a transient trapdoor scheme, a novel revocable fast attribute-based encryption scheme, and a dynamic group signature scheme. In the same year, Xu et al. [16] proposed a round-based redactable blockchain scheme based on chameleon hash function and digital signature technology, which stipulated that the modifier could perform up to K data edits in each round. Once the limit was exceeded, anyone could trace the source by extracting the malicious modifier's private key. However, this scheme did not consider the limitation of data editing frequency. Once the data editing frequency was too fast, it would trigger serious DDoS attacks.

In 2022, Ma et al. [17] proposed a new cryptographic primitive called distributed attribute-based chameleon hash (DPCH) based on the PCH scheme to address the above-mentioned issues. The modifier's permissions are issued by multiple authoritative institutions, so there is no need for fully trusted authoritative institutions. At the same time, DPCH can resist collusion attacks between authoritative institutions and modifiers, and there is no need for any interaction between various authoritative institutions. However, there are too many cryptographic primitives and the computation is complex.

In 2023, Li et al. [18] proposed a non-interactive chameleon hash (NITCH) scheme, which only requires collecting enough subkeys when users need to modify data. Li also designed a periodic update committee scheme in the article, which can effectively solve the damage caused by key destruction and ensure the anonymity of committee members. However, the voting stage during the modification process takes a long time. Chen Yue et al. [19] proposed a chameleon hash function with editing limit, which includes a master trapdoor, a slave trapdoor, and a generated witness. The slave trapdoor is used to edit data, the master trapdoor is used to revoke the slave trapdoor to achieve the purpose of revoking editing permissions, and the witness can strictly limit the editing limit of the slave trapdoor to once, but the revocation of the trapdoor lacks flexibility.

In 2024, Zhao et al. [20] proposed Tiger Tally as a secure redactable architecture, introducing a novel targeted policy-based chameleon hash algorithm and tokenized editing permissions to form an integrated cryptographic and access control mechanism. However, tokenized editing permission management may bring new security risks, such as the risk of token theft or abuse.

In order to achieve an efficient decentralized blockchain redactable scheme, this paper introduces the chameleon Hash function to construct a redactable blockchain and designs a chameleon hash with dynamic trapdoor (DTCH) scheme that supports dynamic trapdoors. This scheme can dynamically update trapdoors when breaking the inherent immutability of the blockchain, thereby reducing the risk of malicious nodes holding trapdoors for a long time. Specifically, the trapdoor key will automatically update after a certain period of time or after being used once. In this case, the authorized node cannot successfully find the hash collision using the old key, thus effectively revoking its authorization. One major advantage of this method is that even if the key is stolen, the key held by the thief will expire after a period of time.

3. Preliminaries

In this section, we introduce the preliminary knowledge used throughout the paper and give the security requirements that DTCH needs to satisfy.

3.1. Bilinear map

Bilinear Map is a mathematical structure widely used in cryptography, especially in designing secure protocols and constructing encryption algorithms. The following are the basic concepts and properties of bilinear map:

Definition 1: Given two multiplicative cyclic groups G and G_T of order q, g represents the generator of group G. The bilinear map e from G to G_T: $G \times G \to G_T$ satisfies the following properties:

(1) Bilinear: For any $x, y \in G$ and any $a, b \in Z_q$, $e(x^a, y^b) = e(x, y)^{ab}$ always holds.
(2) Nondegeneracy: Any $x, y \in G$, such that $e = (x, y) \neq 1_{G_T}$ holds, where 1_{G_T} is the unit element of the multiplicative cyclic group G_T.
(3) Computability: For any $x, y \in G$, there exists an effective algorithm to compute $e = (x, y)$ in polynomial time. The following presents a computability difficulty problem defined on group G.

Definition 2: Discrete Logarithm Problem (DLP) Given a binary (g, g^a), the output $a \in Z_q$, where g is the generator of the multiplication loop group G. If there exists a negligible function $negl(\lambda)$ for any security parameter λ and any probability polynomial-time adversary $\mathcal{A}$, such that:

$$Pr\left[\mathcal{A}(1^\lambda, g, g^a) = a : a \overset{R}{\leftarrow} Z_q\right] \leq negl(\lambda)$$

Then it is said that the DLP assumption on group G holds true.

3.2. Chameleon hash function

Hash functions typically have collision resistance, which means that when given two different inputs, their outputs must also be different. Moreover, even with small changes in input data, there will still be significant differences in output, which is the key guarantee of blockchain's immutability. The chameleon hash function is a category of hash functions, proposed by Krawczyk and Rabin as a single item hash function with trapdoors [21, 22]. The algorithm consists of four parts: initialization, key generation, hash calculation, and collision generation.

Chameleon hashing, on the premise of implementing hashing, can pre-set a backdoor called a private key or trapdoor, which can easily identify collisions (making m and m' hash values the same, where m equals m'). The chameleon hash function can be defined as: given the chameleon hash public key hk, arbitrary data m, and randomly selected parameters r, if the chameleon hash trapdoor tk is unknown, the chameleon hash $CH(m, r, hk)$ has the unidirectionality and collision resistance of general hash functions; If you hold the chameleon hash trapdoor tk, you can easily find the corresponding message (m, r) (m', r'), making the chameleon hash $CH(m, r, hk) = CH(m', r', hk)$. Although this breaks the two collision-resistance of the hash function mentioned above, for most people, these characteristics still exist. Therefore, this function is still safe.

A standard chameleon hash algorithm $CH = (Setup, KeyGen, Hash, Forge)$. It usually includes four parts: initialization, key generation, hash calculation, and hash collision. The specific description is divided into the following four steps.

(1) Parameter generation algorithm $CH.Setup(\lambda) \to pp$: Input the security parameter λ and output the common parameter p.
(2) Key generation algorithm $CH.KeyGen(pp) \to (hk, sk)$: Input the public parameter pp and output the chameleon hash's public key hk and private key sk (also known as trapdoors).
(3) Hash calculation algorithm $CH.Hash(hk, m, r) \to (h)$: Input the public key hk, message m, random number r, and output the chameleon hash value h.
(4) Hash collision algorithm $CH.Forge(sk, m, m', r) \to (r')$: Input the private key sk, two different messages m, m', random number r, output a new random number r', and satisfy $CH(sk, m, r) = CH(sk, m', r')$.

3.3. Security requirements

The security requirements of a secure chameleon hashing scheme are mainly based on the security model given in reference [13], which includes the security requirements of correctness, revocability, and collision resistance.

Correctness: A chameleon hash CH is called correct, if for all security parameter $\lambda \in \mathbb{N}$, for all $CH.Setup(\lambda) \to pp$, for all $CH.KeyGen(pp) \to (hk, sk)$, for all m', we have for all $CH.Forge(sk, m, m', r) \to (r')$, that $CH(sk, m, r) = CH(sk, m', r')$ is always true.
Revocability: Revocability requires the ability of a data modifier to find a collision of a certain hash value after one edit of a pre-given message under a certain hash value has occurred by the data modifier. A DTCH is revocable, if for all efficient adversary $\mathcal{A}$ it holds that $Pr\left[Expt_{\mathcal{A}}^{Revoke}\lambda = 1\right] \leq negl(\lambda)$
Collision resistance: A chameleon hash CH is enhanced collision-resistant, if for all probabilistic polynomial-time adversary $\mathcal{A}$, the following probability is negligible: $Pr[(m_1, m_2) \leftarrow \mathcal{A}(1^\lambda, CH) : m_1 \neq m_2 \wedge CH(m_1) = CH(m_2)]$.

4. Scheme Design

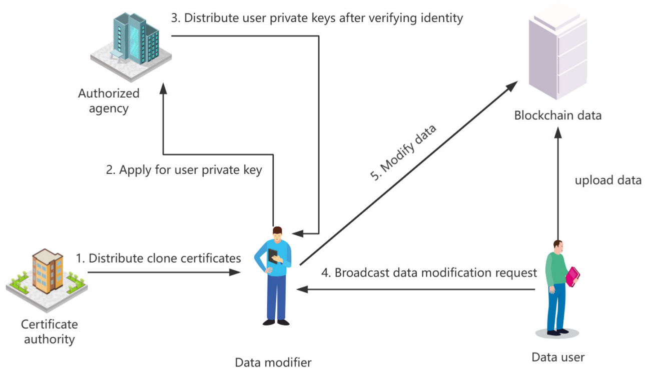
In this section, we present the system model and the concrete construction of DTCH.

4.1. System model

As shown in Figure 1, the scheme in this article mainly involves four entities: Certificate authority, authorized agency, data user, and data modifier.

1) Certificate authority: The Certificate authority is responsible for initializing the entire system and distributing identity certificates to data modifiers.
2) Authorized agency: After verifying the identity of the data modifier, the authorized agency is responsible for distributing user keys (from trapdoors) to the data modifier, ensuring that blockchain data can only be edited if the editing policy is met.
3) Data user: The data user is responsible for packaging the raw data into a transaction form and broadcasting it to the blockchain network, and broadcasting data modification requests when there is a need to modify the relevant data.

Figure 1
System model


3. Distribute user private keys after verifying identity
Authorized agency
Blockchain data
5. Modify data
2. Apply for user private key
upload data
1. Distribute clone certificates
4. Broadcast data modification request
Certificate authority
Data modifier
Data user

4) Data modifier: When the attribute set owned by the data modifier meets the corresponding editing strategy, it can initiate editing of blockchain data.

Below is the workflow of the proposed scheme in this article. The certificate authority first initializes the entire system and broadcasts the generated public parameters to other entities in the system, while the data modifier completes the generation of its own user key. Subsequently, the data user broadcasts the transaction to the blockchain network and adds the transaction to the blockchain. If illegal data is found in a transaction on the blockchain, the data user can initiate an editing request to the data modifier. After verifying the legality of the editing request, the data modifier updates the corresponding transaction content on the blockchain.

4.2. Concrete construction

This article mainly includes the following polynomial time algorithms: parameter generation algorithm *CHInit*, trapdoor generation algorithm *TrapdoorGen*, identity-based user key generation algorithm *ModKGen*, hash generation algorithm *CHGen*, collision search algorithm *FindCollision*, hash verification algorithm *VerifyCollision*, trapdoor update algorithm *UpdateEtd*, and update verification algorithm *UpdateVerify*.

1) $CHInit(\lambda) \rightarrow (pp)$: The algorithm is executed by the authorized authority. Input the selected security parameter λ, select the multiplication loop group G and G_T with order q and generator g, choose the hash function $H : \{0,1\}^* \rightarrow G, H_1 : \{0,1\}^* \rightarrow Z_q$, and set the bilinear mapping $e : G \times G \rightarrow G_T$. Output the common parameters $pp\{G, G_T, e, g, H, H_1\}$ and use them as implicit inputs for all other algorithms.

2) $TrapdoorGen(pp) \rightarrow (mtk, hk)$: The algorithm is executed by authorized agencies. Input the public parameter *pp*, generate the system master trapdoor key *mtk* and the system master public key *hk*, and output the chameleon hash master trapdoor key pair (mtk, hk). The system master trapdoor key *mtk* is kept by an authorized institution and the system master public key *hk* is made public. The process of generating the main trapdoor key pair is as follows:

Select the random number $\mathrm{x} \in \mathrm{Z}_\mathrm{q}^*$, calculate $y = g^x$, run the algorithm to generate the main trapdoor key pair (mtk, hk), and make the system's main public key $hk = y$ and main trapdoor key $\mathrm{mtk} = \mathrm{x}$.

3) $ModKGen((mtk, hk), CID, n, h_{valid}, ctime()) \rightarrow (ID_{pp}, tk)$: The algorithm is executed by the authorized authority. The data modifier sends its own identification *CID* to the authorization agency, requesting the generation of a user key *tk*. The authorized agency uses the main trapdoor key *mtk* to calculate the user private key *tk* corresponding to the identity identifier *CID*, inputs the main trapdoor key pair (mtk, hk), identity identifier *CID*, maximum allowed modification times *n*, user key usage period h_{valid}, returns the timestamp function $ctime()$, generates specific identity identifiers for all data modifiers, and calculates *h*:

$$h = H\left(CID|| \left[\frac{ctime()}{h_{valid}}\right]\right), \tag{1}$$

Calculate the user key $tk = h^x$. Output the identity parameters of the data modifier $ID_{PP}\{CID, h_{valid}, h\}$ and the user key *tk*.

4) $CHGen(hk, ID_{PP}, m, r) \rightarrow (CH)$: The algorithm is executed by the data modifier. Enter the system master public key *hk*, the data modifier identity parameter ID_{PP}, the original information string $m \in \{0,1\}^*$, and a random number *r*, and output the hash value *CH*. The process of calculating the hash value *CH* corresponding to information *m* is as follows:

Randomly select the integer α, calculate $r = g^{\alpha x}$, and calculate:

$$CH = e\left(hk, h^{H_1(m)}\right) \cdot e(g, r) \tag{2}$$

5) $FindCollision(hk, (CH, m, r), m') \rightarrow \left(CH', r'\right)$: The algorithm is executed by the data modifier. Enter the system master public key *hk*, chameleon hash value *CH*, original message string *m*, random number *r*, new message string m', and calculate the hash value CH' corresponding to m'. The calculation process is as follows:

① Calculate the temporary proof $tw' = tk' \cdot \mathrm{tx}$, where *tx* refers to the specific operation request.
② Calculate and save $etd' = etd^{tw'}$;
③ Obtain the random number *r*, message *m*, public key *hk*, *h*. Calculate $hk' = y' = y^{tw'}$,

$$\begin{aligned} r' &= g^{\alpha' x'} \\ &= \frac{etd^{H_1(m)}(g^{\alpha x})}{etd'^{H_1(m')}} = \frac{etd^{H_1(m)}(g^{\alpha x})}{etd^{H_1(m')tk'tx}} \\ &= g^{\alpha x} etd^{H_1(m)-H_1(m')tk'tx} \\ CH' &= e\left(hk', h^{H_1(m')}\right) \cdot e(g, r') \end{aligned} \tag{3}$$

Output hash value CH' and random number r'.

(1) *VerifyCollision* $\left(hk, (CH, m, r), \left(CH', m', r'\right), ID_{PP}, tk\right) \rightarrow Status$: The algorithm is executed by the data modifier. Enter the system master public key *hk*, (CH, m, r), $\left(CH', m', r'\right)$, data modifier identity parameter ID_{PP}, and user key *tk*. The specific execution of this algorithm is as follows:

① Calculate whether $e\left(y, (h^{tx})^{tk'}\right)$ is equal to $e\left(y', h\right)$, and if not, output $Status = False$.
② Obtain the modifier ID *CID* and user key *tk* parameters from the data modifier identity parameter ID_{PP}, and determine whether the user private key is within its validity period and has enough editing times. If it is not within its validity period or does not have enough editing times, output $Status = Expired$.
③ Retrieve *CH* and CH', determine if $CH = CH'$ holds, output $Status = Invalid$ if yes, otherwise output $Status = Valid$.

(2) $UpdateEtd(ID_{PP}, h_{valid}, ctime()) \rightarrow \left(mtk', hk', tk'\right)$: The algorithm is executed by an authorized agency. Abolish the old trapdoor and select the new trapdoor. The updated trapdoor satisfies the new trapdoor value that does not depend on the old trapdoor value. The calculation process is as follows:

① Based on the input data, modify the identity parameter ID_{PP} to obtain the current (CH, m, r) and user key $tk = h^x$.
② Call and execute the *TrapdoorGen* algorithm, select a random integer x', calculate parameter $y' = \mathrm{g}^{x'}$, and output a new chameleon hash master trapdoor key pair (mtk', hk').
③ Call and execute the *ModKGen* algorithm. The authorization authority uses the main trapdoor key mtk' to calculate the user key *tk* corresponding to the identity identifier *CID*. The authorization authority generates specific identity identifiers for all data modifiers, calculates h', and the user key $tk' = \mathrm{h}'^{x'}$.

(3) $UpdateVerify((tk, hk), (CH, m, r)) \rightarrow Status$: The algorithm is executed by the authorized agency. Execute collision search algorithm *CollisionFind* and hash verification algorithm *CollisionVerify*. If *tk* is invalid or $CH \neq CH'$, output $Status = Success$, that is, the old trapdoor cannot generate a valid hash value, and the verification is revoked successfully. Otherwise, output $Status = Fail$.

5. Security Analysis

In this section, we give a detailed security analysis of DTCH, which includes correctness, revocability, and collision resistance.

5.1. Correctness

The correctness of this scheme depends on the correctness of the cryptographic primitives commonly used in algorithm construction, as demonstrated below.

① *FindCollision*:

$$
\begin{aligned}
&CH' \\
&= e\left(y', h^{H_1(m')}\right) \cdot e(g, R') \\
&= e\left(g^{x'}, h^{H_1(m')}\right) \cdot e\left(g, \frac{etd^{H_1(m)} g^{\alpha x}}{etd'^{H_1(m')}}\right) \\
&= e\left(g^{x'}, h^{H_1(m')}\right) \cdot e\left(g, \frac{h^{x H_1(m)} g^{\alpha x}}{h^{x' H_1(m')}}\right) \\
&= e\left(g, h^{x' H_1(m')}\right) \cdot e(g, g^{\alpha x}) \cdot e\left(g, h^{x H_1(m) - x' H_1(m')}\right) \\
&= e\left(g, h^{x_u H_1(m)}\right) \cdot e(g, g^{\alpha x}) \\
&= e\left(g^{x}, h^{H_1(m)}\right) \cdot e(g, g^{\alpha x}) \\
&= e\left(y, h^{H_1(m)}\right) \cdot e(g, g^{\alpha x}) = CH
\end{aligned}
$$

Confirmed by $CH^{'} = CH$, completed.

② *UpdateEtd*

$$
\begin{aligned}
&CH'' \\
&= e\left(y', h^{H_1(m'')}\right) \cdot e\left(g, g^{\alpha'' x''}\right) \\
&= e\left(g^{x''}, h^{H_1(m')}\right) \cdot e\left(g, h^{(x' - x'') H_1(m')} g^{\alpha' x'}\right) \\
&= e\left(g, h^{x'' H_1(m')}\right) \cdot e\left(g, h^{(x' - x'') H_1(m')}\right) \cdot e\left(g, g^{\alpha' x'}\right) \\
&= e\left(g^{x'}, h^{H_1(m')}\right) \cdot e\left(g, g^{\alpha' x'}\right) = CH'
\end{aligned}
$$

Confirmed by $CH^{''} = CH'$, completed.

5.2. Revocability

Lemma 1: If H is a collision-resistant hash function and G is a secure pseudo-random number generator, then for the same message *m* and identity identifier CID, the root hash value CH_M of Merkle tree M must be different.

Proof: Assuming $n, u \in \{0, 1\}^n$, a pseudo-random number generator $G : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ is used to generate random numbers r_1,r_2, i.e., $G(u) \rightarrow r_1 || r_2$. Due to G being a secure pseudo-random number generator, the generated r_1 and r_2 are random and unpredictable. The structure of a Merkle tree is as follows: the leaf nodes of M are in the order of r_1, r_2, $H_1(m)$, and *CID* from left to right. The middle node 1 is $H(r_1 || r_2)$, and the middle node 2 is $H(H_1(m) || CID)$. The root hash value CH_M is the hash result of these nodes. Although the input *m* and *CID* are the same, the combination $H(r_1 || r_2)$ of r_1 and r_2 is also different because they are generated randomly and differently each time, which leads to different root hash values CH_M. If there are two identical root hash values CH_M, it means that different inputs (r_1, r_2) generated the same hash value, which violates the collision resistance of the hash function H.

Theorem 1: Assuming that hash functions H_1 and H have collision-resistant properties, and G is a secure pseudo-random number generator, then the system has revocability. During the proof process, this will be verified through a game between challenger $\mathcal{C}$, adversary $\mathcal{A}$, and adversary $\mathcal{B}$.

Proof: Assuming there exists a polynomial-time adversary $\mathcal{A}$ that can effectively attack revocability, construct adversary $\mathcal{B}$ to use the result of $\mathcal{A}$ to attack the collision resistance of hash function H. In this setting, $\mathcal{A}$ is allowed to perform multiple $\mathcal{O}_{update}$ queries, with a frequency of $q \in poly(\lambda)$.

① Setting stage: Challenger $\mathcal{C}$ first provides adversary $\mathcal{B}$ with a collision-resistant hash function $H_1 : \{0, 1\}^n \rightarrow G$. Then, $\mathcal{B}$ constructs DTCH through H_1 and initializes parameters according to the requirements of revocability experiments.

② Query phase: Allow $\mathcal{A}$ to perform an $\mathcal{O}_{update}$ query and provide feedback to $\mathcal{A}$ on the output result (r_i, tk_i) of the *i*-th query, where $i \in [0, q]$. Additionally, maintain a table $\mathcal{Q}$ to store tk_i.

③ Challenge stage: When $\mathcal{A}$ outputs $(tk, (CH, m, r), m')$ to $\mathcal{B}$, $\mathcal{B}$ checks the validity of (CH, m, r) and verifies whether *tk* is the user key of (CH, m, r), querying whether it exists in table $\mathcal{Q}$. If the verification is successful, $\mathcal{B}$ will run the *UpdateEtd* algorithm and input (CH, m, r) and m' to generate a new user key tk'. Among them, $tk = h^x = H(CID || [ctime()/h_{valid}])^x$ and $r = g^{\alpha x}$. Verify the correctness of the input through the verification function $VerifyCollision(hk, CH, m, r)$. At the same time, verify the equation $e\left(g, tk'^{H_1(m)}\right) = e\left(g^{x'}, H(CID || CH_M)^{H_1(m)}\right)$ to confirm whether *tk* is the user key for $(CH, m,)$. At the same time as generating a new user key tk', $\mathcal{B}$ will construct a Merkle tree M', where the left leaf node is the random number generated by the pseudo-random number generator G, the two nodes on the right are $H_1(m')$ and the custom identity CID', the root hash value is CH'_M, and the user key $tk' = \mathrm{h}'^{x'}$.

④ Finally, $\mathcal{B}$ outputs $H\left(CID^{'} || CH'_M\right)$ and $H(CID || CH_M)$ to challenger $\mathcal{C}$.

If $\mathcal{A}$ has a significantly higher probability of winning in a challenge than $\mathcal{B}$, then the probabilities of the new and old trapdoors are equal in that challenge. Here, *CID* and $CID^{'}$ can be equal, and $\mathcal{A}$ can use the previously queried m' in the challenge. However, according to Lemma 1, the two hash values $H\left(CID^{'} || CH'_M\right)$ and $H(CID || CH_M)$ obtained by $\mathcal{B}$ are different, so the same *m* and *CID* produce different CH_M. Therefore, $CID^{'} || CH'_M$ is not equal to $H(CID || CH_M)$, meaning that $\mathcal{B}$ successfully destroyed the collision resistance of H_1 and won the challenge. Therefore, $\mathcal{B}$ and $\mathcal{A}$ have the same advantage in winning this game.

5.3. Collision resistance

Theorem 2: If there exists an effective adversary $\mathcal{A}$ that can successfully find a collision of chameleon hash functions, construct an adversary $\mathcal{B}$ to solve the problem of computing discrete logarithm (CDH). Specifically, adversary $\mathcal{A}$ is capable of generating effective collisions, and adversary $\mathcal{B}$ can utilize this ability to solve the CDH problem.

Proof: Provide an example of a CDH problem $\left(g, g^a, g^b\right)$, construct $\mathcal{B}$, which can use a to calculate g^{ab}. For constructing adversary $\mathcal{B}$ as follows:

① Setting stage: $\mathcal{B}$ runs *CHInit* to generate common parameters, selects information m, runs *ModKGen* and *CHGen* to obtain the identity parameter ID_{pp} and the hash value CH corresponding to m. At this point, $\mathcal{B}$ does not know the specific value of key x, while the public key value y and the α in r are random numbers. $\mathcal{B}$ sends the identity parameters ID_{pp} and CH to $\mathcal{A}$.

② Query stage: $\mathcal{A}$ queries the hash key and chameleon hash CH based on the identity identifier CID and information m. Hash value calculation $CH(m) = e\left(hk, h^{H_1(m)}\right) \cdot e(g, r)$, where r is a randomly selected integer. Adversary $\mathcal{B}$ can request the generation of a user key. In this process, the authorization agency generates the user key tk and identity parameter ID_{pp} using the master key mtk and identity identifier CID.

③ Challenge stage: In the challenge stage, adversary $\mathcal{A}$ proposes two different messages m_1 and m_2 and obtains their corresponding hash values. Adversary $\mathcal{A}$ takes these messages and their hash values as a challenge. Select two messages m_1 and m_2, calculate the hash value: $CH = e\left(hk, h^{H_1(m)}\right) \cdot e(g, r)$, where r is a random number corresponding to m. In the computation phase, the collision found by adversary $\mathcal{A}$ is used to solve the discrete logarithm (CDH) problem. Collision calculation assumes that adversary $\mathcal{A}$ successfully finds a collision, i.e., finds two different messages m_1 and m_2, such that $CH(m_1) = CH(m_2)$. Calculate collision hash value: $e\left(hk, h^{H_1(m_1)}\right) \cdot e(g, r_1) = e\left(hk, h^{H_1(m_2)}\right) \cdot e(g, r_2)$, because r_1 and r_2 are random numbers, they can be simplified as:

$$\frac{e(g,r_1)}{e(g,r_2)} = \frac{e\left(hk,h^{H_1(m_1)}\right)}{e\left(hk,h^{H_1(m_2)}\right)}, \tag{4}$$

$$r = \frac{e\left(hk,h^{H_1(m_2)}\right)}{e\left(hk,h^{H_1(m_1)}\right)} \cdot \frac{e(g,r_2)}{e(g,r_1)} \tag{5}$$

Use the above calculation results to solve the CDH problem: calculate g^{xy} using r and the master key x, which belongs to solving the CDH problem.

(4) If adversary $\mathcal{A}$ can successfully find a collision, even if $CH(m_1) = CH(m_2)$, then adversary $\mathcal{B}$ can use this collision to solve the discrete logarithm problem.

Due to the difficulty of the CDH problem, the collision resistance of the chameleon hash function is also effective. Through the above process, it has been proven that the collision resistance of the chameleon hash function depends on the difficulty of the CDH assumption. If the CDH problem is unsolvable under given security parameters, then the chameleon hash function will also be collision-resistant.

6. Performance Evaluation

In this section, the theoretical analysis of DTCH is carried out to evaluate the function and computational complexity of the scheme. Then through experimental simulation and research comparison, we present the computation time of the scheme with experiments, and compared with the computation cost of each algorithm of other related schemes to evaluate the efficiency of DTCH.

6.1. Theoretical analysis

6.1.1. Functionality comparison

In terms of functionality, this article compares DTCH with other related schemes and analyzes the performance advantages of DTCH. Table 1 shows the functional comparison between this scheme and various schemes. It can be seen from Table 1 that TCH [11] uses verifiable secret sharing technology to distribute chameleon hash trapdoor shares to other nodes, but this scheme relies on a trusted third party to distribute chameleon hash keys, which belongs to a centralized redactable blockchain scheme. Although RCH [12] implements trapdoor revocation and trapdoor updatability, the modification mechanism is not flexible enough and there is trapdoor exposure. KERB's [16] scheme has no trapdoor exposure problem, but it fails to realize trapdoor revocation and trapdoor updatability, and cannot solve the problem of loss of editing rights after key destruction. RPCH [14] scheme, ReTRACe [15] scheme, and RCHLR [19] scheme have no trapdoor exposure problem and can realize trapdoor revocation. However, trapdoors cannot be updated and are not flexible enough to respond to changing requirements, and may not adapt to new situations or new business requirements, thus affecting their long-term availability. However, the scheme proposed in this paper has no trapdoor exposure problem and has both trapdoor revocation and trapdoor updating functions. The scheme proposed in this paper can meet all expected characteristics.

Table 1
Functionality comparison of different schemes

Schemes	No key exposure problem	Revocation of trapdoor	Trapdoor can be updated
DTCH	√	√	√
TCH [11]	×	×	×
RCH [12]	×	√	√
RPCH [14]	√	√	×
ReTRACe [15]	√	√	×
KERB [16]	√	×	×
RCHLR [19]	√	√	×

6.1.2. Computational complexity

Because RPCH [14] and KERB [16] are based on ABE implementation, their algorithms involve a large number of group operations and bilinear pairing operations, while ReTRACe [15] does not provide concrete construction, these three algorithms are ignored in comparison. Compare the main algorithms of DTCH with the main algorithms in TCH [11], RCH [12], and RCHLR [19] schemes in Table 1. The focus of the comparison is on the number of complex operations, where T_e represents group exponentiation, T_m represents group multiplication, and T_p represents bilinear pairing operation. Due to the fact that the trapdoors in TCH are divided into k parts, the computational cost of this scheme will depend on the parameter k. Table 2 shows that the DTCH scheme proposed in this paper requires slightly more operations of group exponentiation on the *CHGen* algorithm but has fewer operations of group exponentiation and group multiplication on the *FindCollision* and *UpdateEtd* algorithms. In the *VerifyCollision* algorithm, there is a user key expiration verification operation, which improves security and does not generate too many group exponentiation and bilinear pairing operations. Overall, the computational complexity of this scheme is relatively low.

6.2. Experimental results and analyses

6.2.1. Simulation environment

This article conducted real experiments to test the efficiency of the proposed chameleon hash scheme. The experiment was conducted on laptops equipped with PBC library (version 0.5.14) and *openssl* library (version 1.1.1). The device is configured with

Table 2
Computational complexity of different algorithms

	Algorithms			
Schemes	*CHGen*	*FindCollision*	*VerifyCollision*	*UpdateEtd*
DTCH	$4T_e + T_m$+$2T_P$	$2T_e + T_m$	$2T_e + T_m$+$3T_P$	$3T_e + 3T_m$
TCH [11]	$3T_e + T_m$	$(k+1)T_e + 3T_m$+$2T_P$	$T_e + T_m$+$2T_P$	N/A
RCH [12]	$5T_e + 2T_m$+$2T_P$	$2T_e + 2T_m$	$2T_e + T_m$+$4T_P$	$8T_e + 7T_m$
RCHLR [19]	$3T_e + 2T_m$+$3T_P$	$2T_e + 3T_m$+$3T_P$	$T_e + 2T_m$+$3T_P$	$3T_e + 4T_m$+$3T_P$

Figure 2
Computation cost of algorithms for DTCH

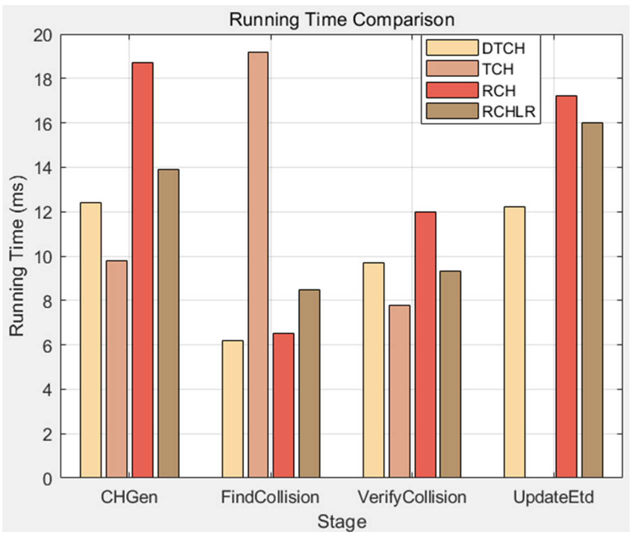
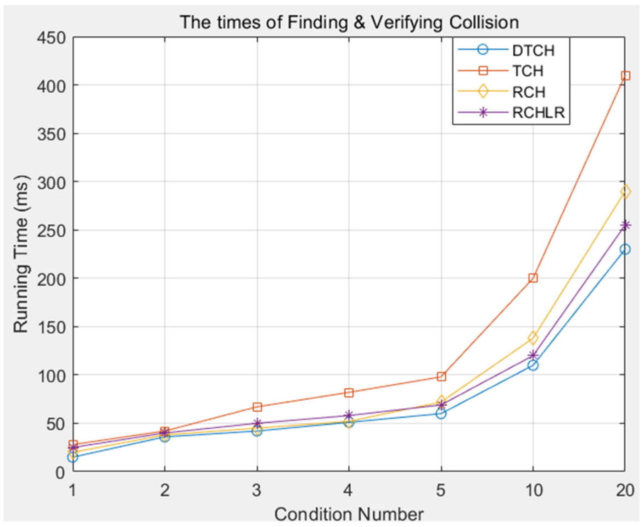


Figure 3
Computation cost of key algorithms for different schemes



a 1.6-GHz 4-core CPU and 4GB RAM, using Ubuntu 18.04 LTS (64 bit) operating system.

6.2.2. Computation cost

The experimental results are provided here to further analyze and compare the computation cost of different schemes. Figure 2 shows the actual computation cost of the main algorithms in the constructed DTCH scheme, compared with the TCH [11] scheme and RCH [12] scheme in Table 1 on *CHGen*, *FindCollision*, *VerifyCollision*, and *UpdateEtd* algorithms, using a messagem size of 1KB. In order to prevent trapdoor owners from abusing trapdoors, TCH divides trapdoors into k parts and hands them over to k trusted organizations. In this experiment, we set k to a small value ($k = 10$) because TCH requires a lot of exponential calculations, and the larger k, the worse the performance of TCH.

As shown in Figure 2, in the *UpdateEtd* algorithm, compared with RCH and RCHLR [19], the DTCH scheme has the shortest running time and the best efficiency; In the *FindCollision* algorithm, this paper has a smaller time cost compared to its proposed schemes, and is significantly more efficient than TCH; In *CHGen* and *VerifyCollision* algorithms, there is also a smaller time overhead.

Because in practical applications, the frequent execution of *FindCollision* and *VerifyCollision* can better reflect the efficiency of the chameleon hash scheme. This experiment will execute *FindCollision* and *VerifyCollision* multiple times and combine their computation cost. Figure 3 shows the computation cost of different schemes after multiple runs. The message used in this experiment is randomly generated and has a size of 1KB. The value of k in TCH remains at 10.

The trapdoor in TCH cannot be updated, and the problem of trapdoor abuse still exists. RCH has implemented regular updates of trapdoors, but the misuse of trapdoors within their validity period is still inevitable. RCHLR has implemented trapdoor undo, but trapdoor undo can only be done after a certain number of edits, which lacks flexibility. Our scheme does not allow the reuse of the same trapdoor, which fundamentally solves the problem of trapdoor abuse and allows for flexible updating of trapdoors. At the same time, compared to other schemes, it does not incur excessive computation cost. This article imposes strict restrictions on the use of trapdoors without affecting the performance of the scheme.

7. Discussion

In order to solve the contradiction between the immutability of data on the blockchain and data supervision, scholars have proposed the concept of redactable blockchain, and it has been continuously studied and developed in recent years. Although Chameleon hash function provides technical support for redactable blockchain, there are security risks of trapdoor abuse and trapdoor leakage in existing redactable blockchain schemes [23]. In this paper, chameleon hash function is introduced to construct redactable blockchain, and a redactable blockchain scheme DTCH is designed. The system model and concrete construction of the proposed scheme are given, and the chameleon hash with dynamic trapdoor is used to restrict the editing permission of data modifier. For the proposed scheme, a detailed security analysis was given, and the functional characteristics and computational overhead of the scheme were evaluated through experimental simulation and research, which proved the effectiveness and feasibility of the scheme. We believe that the proposed scheme is a very promising solution that can be extended to many blockchain systems for efficient and secure transaction rewriting, providing a valuable resource for academics and practitioners.

The proposed scheme still has further research significance. The single data modification in this scheme lacks diverse perspectives and experiences, which may lead to one-sidedness and limitations in decision-making and fail to fully consider the needs of different stakeholders. Therefore, when extending to the blockchain application system, designing a multi-party data modification mechanism is also a key research direction.

8. Conclusion

This paper elaborates on the background of redactable blockchain schemes based on chameleon hashing and analyzes the security issues in existing schemes. The advantage of blockchain is data tamper-proof, and editability seems to break this advantage, which is easy to cause data trust problems. Therefore, redactable blockchains that support data modification need to establish secure and perfect trust and supervision mechanisms to prevent the abuse of editing permissions. In this paper, we propose DTCH, a dynamic trapdoor redactable blockchain scheme, and give a concrete construction with provable security based on bilinear map.

Although the existing schemes solve some difficult problems faced by redactable blockchain, they still do not establish a secure and reliable trust environment, and the modification mechanism is not flexible enough to solve the problem of loss of editing rights after key destruction. In addition, the existing redactable blockchain schemes that can realize trapdoor retracement are not flexible enough to respond to changing requirements because the trapdoor cannot be updated. It may not be able to adapt to new situations or new business requirements, thus affecting issues such as its long-term availability. The DTCH proposed in this paper utilizes controllable randomness, allowing only users with valid trapdoor keys to make changes to the uploaded data, and the user keys are time-sensitive. When the user key expires or is used, the system will generate new trapdoors to abolish the old ones, reducing the risk of editing permission abuse and effectively solving security issues after trapdoor leakage. Security analysis and related experimental results show that compared with other existing schemes, the proposed scheme can manage the trapdoor in a fine-grained manner, realize the flexible update of trapdoor, and have good efficiency in each stage, which achieves higher security guarantee with a smaller computational cost.

Looking into the future, considering that there is only one data modifier in this paper and the editing power is too centralized, in the next work, the trapdoor key will be fragmented and distributed to multiple data modifiers. When an editing request occurs, the trapdoor key can be reconstructed to edit the blockchain only when the threshold is reached.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Shuqiao Li: Conceptualization, Methodology, Software, Validation, Formal analysis, Formal analysis, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration. **Xiaoming Hu:** Conceptualization, Methodology, Resources, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration. **Yan Liu:** Formal analysis, Formal analysis, Resources, Writing – review & editing, Visualization, Supervision. **Shuangjie Bai:** Validation, Formal analysis, Writing – review & editing, Visualization.

References

[1] Nakamoto, S. (2008). *A peer-to-peer electronic cash system*. Retrieved from: https://www.klausnordby.com/bitcoin/Bitcoin_Whitepaper_Document_HD.pdf

[2] Ismail, L., & Materwala, H. (2019). A review of blockchain architecture and consensus protocols: Use cases, challenges, and solutions. *Symmetry*, *11*(10), 1198.

[3] Zhang, X., Xu, F., & Qin, W. (2023). The important value and application of blockchain in the field of network security [qū kuài liàn zài wǎng luò ān quán lǐng yù dě zhòng yào jià zhí yǔ yìng yòng]. *Journal of the Chinese Academy of Electronic Sciences*, *18*(06), 566–572.

[4] Zhang, C. C., Li, L. X., Du, J. Z., & Shi, J. P. (2024). A review of redactable blockchain research [kě biān jí qū kuài liàn yán jiū zōng shù]. *Computer Engineering and Applications*, *60*(18), 32–49.

[5] Luo, B., Wen, J. M., Mu, Y. D., & Chen, J. (2023). State of the art and challenges of redactable blockchain [kě biān jí qū kuài liàn de yán jiū xiàn zhuàng yǔ tiǎo zhàn]. *Journal of Cyber Security*, *8*(4), 62–84.

[6] Zhang, D., Le, J., Lei, X., Xiang, T., & Liao, X. (2021). Exploring the redaction mechanisms of mutable blockchains: A comprehensive survey. *International Journal of Intelligent Systems*, *36*(9), 5051–5084.

[7] Gao, W., Chen, L. Q., Tang, C. M., & Li, F. (2021). One-time chameleon hash function and its application in redactable blockchain [yí cì biàn sè lóng hā xī hán shù jí qí zài kě xiū zhèng qū kuài liàn zhōng de yìng yòng]. *Journal of Computer Research and Development*, *58*(10), 2310–2318.

[8] Politou, E., Casino, F., Alepis, E., & Patsakis, C. (2019). Blockchain mutability: Challenges and proposed solutions. *IEEE Transactions on Emerging Topics in Computing*, *9*(4), 1972–1986.

[9] Ashritha, K., Sindhu, M., & Lakshmy, K. V. (2019). Redactable blockchain using enhanced chameleon hash function. In *2019 5th International Conference on Advanced Computing & Communication Systems,* 323–328.

[10] Derler, D., Samelin, K., Slamanig, D., & Striecks, C. (2019). Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based. *Cryptology ePrint Archive*.

[11] Huang, K., Zhang, X., Mu, Y., Wang, X., Yang, G., Du, X., . . . , & Guizani, M. (2019). Building redactable consortium blockchain for industrial Internet-of-Things. *IEEE Transactions on Industrial Informatics*, *15*(6), 3670–3679.

[12] Huang, K., Zhang, X., Mu, Y., Rezaeibagha, F., Du, X., & Guizani, N. (2019). Achieving intelligent trust-layer for Internet-of-Things via self-redactable blockchain. *IEEE Transactions on Industrial Informatics*, *16*(4), 2677–2686.

[13] Jia, Y., Sun, S. F., Zhang, Y., Liu, Z., & Gu, D. (2021). Redactable blockchain supporting supervision and self-management. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 844–858.

[14] Xu, S., Ning, J., Ma, J., Xu, G., Yuan, J., & Deng, R. H. (2021). Revocable policy-based chameleon hash. In *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security*, 327–347. https://doi.org/10.1007/978-3-030-88418-5_16

[15] Panwar, G., Vishwanathan, R., & Misra, S. (2021). ReTRACe: Revocable and traceable blockchain rewrites using attribute-based cryptosystems. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies,* 103–114. https://doi.org/10.1145/3450569.3463565

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

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

[18] Li, J., Ma, H., Wang, J., Song, Z., Xu, W., & Zhang, R. (2023). Wolverine: A scalable and transaction-consistent redactable permissionless blockchain. *IEEE Transactions on Information Forensics and Security*, *18*, 1653–1666.

[19] Chen, Y., Hao, Z. H., Wei, J. H., & Yang, D. M. (2023). Redactable blockchain supporting trapdoor revocation and limited number of redactions [zhī chí xiàn mén chè xiāo hé biān jí cì shù xiàn zhì de kě biān jí qū kuài liàn]. *Journal on Communications*, *44*(07), 100–113.

[20] Zhao, L., Guo, D., Luo, L., Xie, J., Shen, Y., & Ren, B. (2024). Tiger Tally: A secure IoT data management approach based on redactable blockchain. *Computer Networks*, *248*, 110500.

[21] Krawczyk, H., & Rabin, T. (1998). Chameleon hashing and signatures. *Cryptology ePrint Archive*.

[22] Camenisch, J., Derler, D., Krenn, S., Pöhls, H. C., Samelin, K., & Slamanig, D. (2017). Chameleon-hashes with ephemeral trapdoors: And applications to invisible sanitizable signatures. In *Public-Key Cryptography–PKC 2017: 20th IACR International Conference on Practice and Theory in Public-Key Cryptography*, 152–182.

[23] Zhao, X. Q., Zhang, Z. H., & Li, Y. (2021). An editable and accountable blockchain schemel [kě biān jí qiě kě zhuī zé de qū kuài liàn fāng àn]. *Journal of Information Security*, *7*(5), 19–28.

How to Cite: Li, S., Hu, X., Liu, Y., & Bai, S. (2024). A Dynamic Trapdoor Redactable Blockchain Schemes. *Journal of Data Science and Intelligent Systems*. https://doi.org/10.47852/bonviewJDSIS42024450