Journal of King Saud University - Computer and Information Sciences

DCCV：Dynamic Cross-Chain Decentralized Data Consistency Verification

--Manuscript Draft--

Manuscript Number:	JKSUCIS-D-23-00930
Article Type:	Full Length Article
Keywords:	cross-chain; Dynamic data; smart contract; Data consistency verification; Decentralized chameleon hash function
Abstract:	The continuous increase in the number of transactions is exerting significant pressure on the storage capacity of blockchains. By storing data on multiple blockchains and interacting with cross-chain technology, the storage pressure can be relieved to a large degree. Nevertheless, the feature of immutability may impede the further development of blockchains, e.g., some expired identity information cannot be updated due to immutability. The decentralized chameleon hash function can resolve the issue while ensuring the feature of decentralization. Unfortunately, how to ensure the consistency of dynamic data updating in cross-chain interaction is still a question worth considering. In this paper, we propose a dynamic cross-chain decentralized data consistency verification (DCCV) model, in which we adopt an audit chain to guarantee the consistency of dynamic data updating between the source chain and the target chain. In addition, in order to enhance the relevance of calculations across different smart contracts and improve audit efficiency, we propose a way to develop cross-chain smart contracts collaboratively. Moreover, Cosi protocol and multi-signcryption are utilized to ensure the security and privacy of cross-chain data transmission. Finally, theoretical and experimental analysis demonstrates that DCCV can well achieve dynamic data consistency verification in the process of cross-chain interaction.

DCCV: Dynamic Cross-Chain Decentralized Data Consistency Verification

ARTICLE INFO

Keywords:
Cross-chain
Dynamic data
Smart contract
Data consistency verification
Decentralized chameleon hash function

Abstract

The continuous increase in the number of transactions is exerting significant pressure on the storage capacity of blockchains. By storing data on multiple blockchains and interacting with each other with cross-chain technology, the storage pressure can be relieved to a large degree. Nevertheless, the feature of immutability may impede the further development of blockchains, e.g., some expired identity information cannot be updated due to immutability. The decentralized chameleon hash function can resolve the issue while ensuring the feature of decentralization. Unfortunately, how to ensure the consistency of dynamic data updating in cross-chain interaction is still a question worth considering. In this paper, we propose a dynamic cross-chain decentralized data consistency verification (DCCV) model, in which we adopt an audit chain to guarantee the consistency of dynamic data updating between the source chain and the target chain. In addition, in order to enhance the relevance of calculations across different smart contracts and improve audit efficiency, we propose a way to develop cross-chain smart contracts collaboratively. Moreover, Cosi protocol and multi-signcryption are utilized to ensure the security and privacy of cross-chain data transmission. Finally, theoretical and experimental analysis demonstrates that DCCV can well achieve dynamic data consistency verification in the process of cross-chain interaction.

1. Introduction

The blockchain technology is essentially a combination of distributed systems, computer networks, databases, and distributed ledger techniques. Recently, blockchain is widely used in many fields due to the features of decentralization, transparency, immutability, and anonymity Yang et al. (2018); Xu et al. (2016); Haddad (2022). Nevertheless, the increasing number of transactions has brought huge storage pressure and also has a great impact on the efficiency of querying and computing data on the blockchain Thus, in order to relieve the storage pressure on the blockchain, storing data on different blockchains and connecting these heterogeneous blockchains by using cross-chain technology can well relieve the storage pressure on blockchains.

Currently, the mainstream cross-chain technologies can be classified as follows: notary mechanism Hope-Bailie and Thomas (2016), sidechain/relay chain technology Back et al. (2014), and hash locking Poon and Dryja (2016). By establishing connections between different blockchains, cross-chain technology enables the interaction between heterogeneous blockchains. However, the feature of immutability may raise some problems at times. The necessity of enabling dynamic data updates on the blockchain is paramount from the perspective of economics and security. Failure to enable dynamic data management may result in significant adverse economic impacts and may also expose blockchain systems to a range of security risks Gad et al. (2022); Shrimali and Patel (2022). On the one hand, some blockchain systems have suffered malicious transactions in practice, for example, *The DAO*, which is a famous program built on the Ethereum platform, was attacked by hackers due to the loopholes in its smart contracts. The serious issue was solved by Ethereum hark fork, yet the immutability of blockchain also caused billions of losses for *The DAO* community Atzei et al. (2017). On the other hand, the mechanism of allowing to update transactions is indeed demanded in many blockchain applications. For example, some expired identity information needs to be updated from time to time. Thus, achieving data dynamics is critical to enhance the scalability of blockchain and can make it more widely available. Decentralized chameleon hash function Jia et al. (2022) can well achieve the purpose of updating the transactions on the blockchain. However, how to guarantee the consistency of dynamic data updates during cross-chain interaction is a problem worth considering.

Currently, research on dynamic data audit mainly focuses on cloud storage environments Abiodun et al. (2022). Shen et al. implemented a model to support bulk dynamic data auditing using double-linked information tables and location arrays, which is advantageous in terms of computation and communication costs and can achieve a low audit overhead Shen et al. (2017). He et al. was the first to propose an audit model to support fully dynamic data update, which has a fixed storage cost for each generated audit message He et al. (2019). Patil et al. proposed a dynamic data audit model that can protect the privacy of users, which can support dynamic data audit while satisfying the protection of user privacy Patil and Chaudhari (2018). However, previous studies have mainly used a third-party auditor (TPA) to audit the data stored on the cloud for data owners. However, TPA may be malicious, it may conspire with the cloud to harm the interests of data owners. Besides, the introduction of TPA violates the decentralization feature of blockchain. Meanwhile, TPA mainly audits the data stored in the cloud while we need to audit the consistency of the data during the cross-chain interaction. Thus, the traditional audit model cannot be directly applied to the cross-chain scenario. Nowadays, some researches about the consistency of cross-chain data interaction are underway Zhang et al. (2022); Jiang et al. (2022). Nevertheless, the research above

ORCID(s):

mainly focuses on the consistency verification of static data in cross-chain interaction. How to achieve dynamic cross-chain data consistency verification is a question worth studying.

In this paper, we present a dynamic cross-chain decentralized data consistency verification model (DCCV). DCCV implements a separate blockchain, i.e., an audit chain to audit the consistency of data updating between the source chain and the target chain. It's worth noting that consistency in this paper refers to two aspects:

- The data received on the target chain can keep correctness and integrity with the data stored on the source chain.
- The data updated on the source chain are completely and correctly updated on the target chain.

Specifically, the user nodes of the source chain cooperate to generate the decentralized chameleon hash key and update the transactions. The update request and history records are stored in the source chain for auditing. Upon receiving the data update request, the user nodes of the target chain update the transactions according to the request. Finally, the audit chain audits the consistency of data updating. Besides, we deploy different smart contracts in the source chain, the target chain, and the audit chain to achieve consistency audit of dynamic cross-chain data. Meanwhile, in order to make the consistency audit process more fluent, DCCV deploys the smart contracts among the source chain, the target chain, and the audit chain coordinately. Moreover, DCCV utilizes Cosi protocol and multi-signcryption algorithm to ensure the privacy and security of cross-chain data transmission. In summary, we make the following contributions:

- We propose a consistency audit model for dynamic cross-chain data (DCCV) and build a Merkle hash tree that can be dynamically updated by using a decentralized chameleon hash function. Besides, considering the dynamic join or exit of user nodes on the blockchain, we modify the decentralized chameleon hash function to support threshold data updates.
- We deploy different smart contracts on the audit chain, the target chain, and the source chain in a collaborative way, which improves the efficiency and fluency of consistency audit in cross-chain interaction.
- We introduce Cosi protocol Syta et al. (2016) and multi-signcryption algorithm Sharmila Deva Selvi et al. (2009) to guarantee the security and privacy of cross-chain data transmission.

The remainder of this paper is organized as follows. We review the related work in Section 2 and introduce the preliminary knowledge of DCCV in Section 3. We elaborate DCCV in Section 4 and Section 5, followed by the security analysis in Section 6 and experimentally analyzing DCCV in Section 7. We finally conclude the paper in Section 8.

2. Related Work

2.1. Cross-chain Technologies and Applications

The problem of *data silos* among heterogeneous blockchains has become an important reason to hinder the further development and application of blockchain. The birth of cross-chain technology can help break the barrier and realize the exchange of assets and information among heterogeneous blockchains. At present, the mainstream cross-chain technologies include notary mechanism Hope-Bailie and Thomas (2016), side/relay chain technology Back et al. (2014), and hash locking Poon and Dryja (2016). Besides, various applications based on cross-chain and multi-chain technologies, have emerged and facilitated to expand the applications of blockchain.

In 2016, Tsai et al. Tsai et al. (2016) proposed a new cross-chain mechanism, which is named Beihang Chain. The scalability of blockchain, involved in finance, is realized through the Sharding technology based on Account-Blockchain and TradingBlockchain. In 2019, Li et al. Li et al. (2019) proposed a new cross-chain system (AgentChain), which can be compatible with most current blockchains and performs well in terms of decentralization, efficiency, and security. However, these researches are mainly about cross-chain assets transfer and do not involve the interaction of information. In 2022, Zhang et al. Zhang et al. (2022) proposed a blockchain-enabled decentralized consistency verification for cross-chain calculation (BeDCV), which can achieve the purpose of cross-chain consistency verification, by leveraging paillier homomorphic encryption and a counting bloom filter. Nevertheless, dynamic cross-chain data consistency verification lacks adequate research.

2.2. Dynamic Data Consistency Auditing Schemes

Currently, dynamic data consistency auditing schemes mainly can be divided into two classes.

- *Partially dynamic schemes.* Ateniese et al. Ateniese et al. (2008) proposed a partially dynamic provable data possession (PDP), which supports limited modification, deletion, and appending of file blocks, by leveraging symmetric key cryptography. Wang et al. Wang et al. (2012) proposed a flexible distributed storage integrity audit mechanism, which utilized the homomorphic token and distributed erasure-coded data. These schemes can produce shorter proofs compared with *ADS-based schemes*, while the updating cost is very high.
- *ADS-based schemes.* The combination of homomorphic tags and authenticate data structure (ADS). Erway et al. Erway et al. (2015) proposed a dynamic provable data possession (DPDP) model. DPDP is the extension of traditional provable data possession (PDP) and can support auditing for the stored updating data. Wang et al. Wang et al. (2010) proposed a fully dynamic data updating auditing scheme and extended the scheme to support multiple audit tasks simultaneously. These schemes can realize data updating with

high efficiency, but the cost of data consistency auditing is very high. For example, a consistency auditing proof can be up to 100KB for a typical security level, while the average block size of Ethereum is 0.02M (20.48KB). It means that proof cannot be stored in even an entire block.

These consistency auditing schemes are mainly utilized in the cloud storage environment and cannot be applied to the cross-chain scenario directly. Thus, how to audit dynamic data in the emerging context of cross-chain interaction is still an unexplored problem Deng et al. (2023).

3. Preliminaries

3.1. Decentralized Chameleon Hash Function

The traditional chameleon hash function requires a centralized entity to possess the private key for updating transactions while the feature of decentralization makes the blockchain cannot design such an entity. In this paper, we introduce a decentralized chameleon hash function to update the transactions, which mainly consists of the following procedures:

- DCH.KeyGen$((1^l, 1^n)_{i\in[1,n]}) \rightarrow (PK, sk_i)_{i\in[1,n]}$: each entity U_i takes a certain security parameter l and the number of entities n, and outputs a master public key PK and a share of corresponding private key sk_i.
- DCH.Hash$(PK, msg) \rightarrow (r, h)$: each entity U_i takes a master public key PK and a message msg, and outputs a randomness r and a hash value h.
- DCH.ReHash$(PK, (msg, r)) \rightarrow h$: each entity U_i takes a master public key PK and an original message-randomness pair (msg, r), and outputs a hash value h.
- DCH.Update$((sk_i, PK, (msg, r), msg')_{i\in[1,n]}) \rightarrow r'$: each entity U_i takes its share sk_i, a master public key PK, a message-randomness pair (msg, r), and an updated message msg'. Then, U_i outputs an updated randomness r'.
- DCH.Verify$(PK, (msg, r), (msg', r')) \rightarrow 0/1$: each entity U_i takes a master public key PK, an original message-randomness pair (msg, r), and an updated message-randomness pair (msg', r'). Then, U_i outputs 0/1 to indicate the update (msg', r') is invalid or valid.

3.2. Cosi Signature Protocol

Cosi protocol Syta et al. (2016) is a kind of witness cosignature protocol, which is extensible Syta et al. (2016) and consists of the following procedures:

- KeyGen$(1^l) \rightarrow (pk_i, sk_i)_{i\in[1,n]}$: each entity U_i takes a certain security parameter l and outputs a public-private key pair (pk_i, sk_i), where n is the number of entities.
- ParAgg$(pk_i, v_i) \rightarrow (PK, V)$: take the public key pk_i of each entity and a partial aggregation value v_i, and output a master public key PK and an aggregation value V.
- Sign$(msg, V) \rightarrow (c, r)$: take the message msg and an aggregation value V, and output a signature (c, r).
- Verify$(msg, V, V') \rightarrow 0/1$: take the message msg, an aggregation value V and V', and output 0/1 to indicate the message msg is invalid or valid.

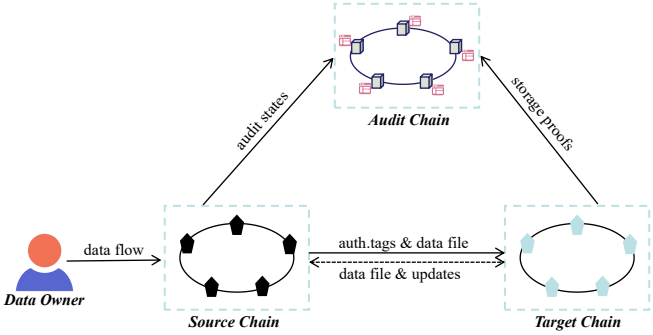

Fig. 1: System Model.

4. Models

4.1. System Model

In this paper, we propose DCCV to implement dynamic data consistency verification in cross-chain interaction. We design three consortium blockchains based on Hyperledger Fabric: a source chain (SC), a target chain (TC), and an audit chain (AC). As is shown in Fig. 1, a data owner (DO) participates in DCCV to process the data, including generating authentication tags and sending data update requests. Specifically, the responsibilities and obligations of the four entities are as follows:

- *Data owner* (DO). DO stores the original data F into the source chain SC. Firstly, DO preprocesses F into blinded data F' and generates authentication tags and a decentralized chameleon hash random value. Then, DO honestly sends F', authentication tags, and decentralized chameleon hash random value to SC for storage.
- *Source chain* (SC). SC receives and honestly stores data F' from DO. When cross-chain data interaction occurs, SC constructs a dynamic Merkle hash tree based on the received data. Then, SC sends authentication tags and F' to TC. When the data F' need to be updated into F^*, SC updates F' and meanwhile broadcasts the update request to AC for consistency auditing and TC for the corresponding

updating. When AC conducts a data consistency challenge, SC generates an audit proof for the data F' and sends it to AC for verification.

- *Target chain* (TC). TC receives the data F' and authentication tags from SC. Besides, TC needs to update F' based on the requirements of SC. When AC conducts a data consistency challenge, TC generates an audit proof based on the data F' and sends it to AC for verification.

- *Audit chain* (AC). AC is an auditing platform, which is set up by the national regulatory authorities. When it conducts a data consistency challenge, AC will receive the audit proofs from SC and TC. After receiving audit proofs, AC implements consistency verification to judge whether the cross-chain interaction is correct.

4.2. Threat Model

In DCCV, AC is responsible to audit dynamic cross-chain interaction consistency between SC and TC. Since AC and the audit smart contract are implemented by national regulatory authorities and the regulatory process of AC is characterized by transparency, traceability, and tamper-proof. Hence, the decentralized feature of blockchain remains uncompromised and AC is considered honest and trustworthy. Besides, DO is mainly involved in processing off-chain original data while we mainly focus on cross-chain data consistency verification, so we assume that DO is honest in DCCV. In summary, there are mainly the following forms of cross-chain attacks:

- *Cross-chain tampering attacks.* The transmitted data F', update requests, and the requirements to construct the smart contracts may be forged or tampered with by an adversary during the process of cross-chain interaction.

- *Cross-chain privacy leakage attacks.* The content of authentication tags and update request may be leaked during the interaction process among SC, TC, and AC.

- *Audit inconsistency attacks.* TC may not accurately retain the received data F' and tags, or update F' as requested. Hence, it may attempt to forge an audit proof to deceive AC.

4.3. Design Goals

Based on the analysis of DCCV, we propose the following design goals to ensure the consistency of dynamic data in cross-chain interaction.

- *The consistency of data.* SC can pass the consistency verification by AC only if the sent data are complete while TC store the sent data completely as well as update the data according to the requirements.

Table 1
Notations and Descriptions

Notations	Descriptions
F	The original data stored on SC
F'	The transferred data to TC
F^*	The updated data by SC
n	The total number of user nodes
t	The threshold number of user nodes
n'	The modified number of user nodes
t'	The modified threshold number of user nodes
ρ	The number of updated transactions
id_{U_i}	The pseudo-identity of the user node U_i
PK	The master public key
SK	The master private key
pk_i	The public key of the user node U_i
sk_i	The private key of the user node U_i

- *Low-overhead supervision.* AC audits the consistency of dynamic cross-chain data by auditing the proofs from SC and TC without obtaining the original data.

- *Data privacy and security.* During the process of cross-chain interaction, we adopt Cosi protocol and multi-signcryption to ensure the privacy and security of the transmitted data.

5. The Proposed Scheme

In this section, we give a detailed description of DCCV about how to audit the consistency of dynamic data in cross-chain interaction. First of all, we describe the overview of DCCV. Then, we present the concrete consistency verification process. Besides, the main notations used in DCCV have been displayed in Table 1.

5.1. Overview

The consistency verification in cross-chain interaction of DCCV includes two aspects.

- AC needs to audit the correctness and integrity of the transmitted data between SC and TC.

- AC needs to audit that TC updates the data completely according to the requirements of SC.

Firstly, DO preprocesses data F into F' and generates authentication tags and a decentralized chameleon hash random value. Then, SC constructs a dynamic Merkle hash tree and transfers F' and authentication tags to TC. The transferred data F' will be encrypted by multi-signcryption to ensure its security and privacy.

Second, SC updates F' into F^* with decentralized chameleon hash function. The newly generated random value will be stored in SC for further consistency verification. Data update requests will be sent to TC and TC updates F' based on the request.

Finally, when AC initiates an audit challenge, TC and SC generate a proof respectively and send them to AC.

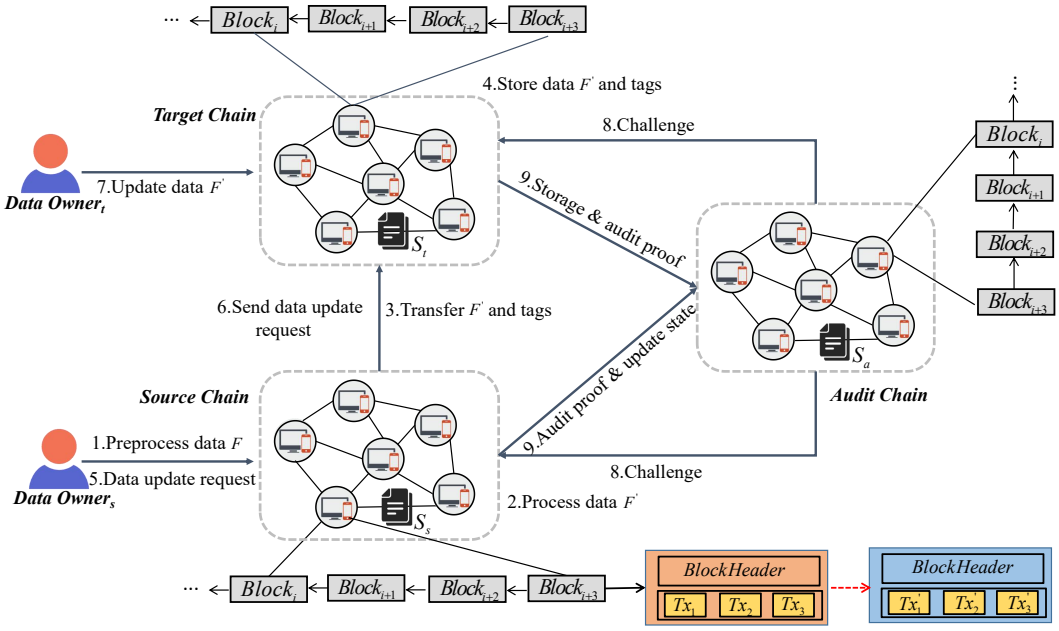

Fig. 2: Overview of the proposed model.

AC verifies the data consistency during the cross-chain interaction. The overview of DCCV is shown in Fig. 2.

5.2. System Initialization

- Upon receiving a security parameter 1^l, $\{U_i\}_{i\in[1,n]}$ select an additive cyclic group G of prime order p, a positive integer domain Z_q^* of prime order q, and three hash functions $H_1 : G\times\{0,1\}^* \to G$, $H_2 : \{0,1\}^*\times Z_q^* \to Z_{q-1}^*$, $H_3 : Z_q^* \to \{0,1\}^l$. Besides, they choose a $(n-1)$-degree polynomial function $f_i(x) = \sum_{j=0}^{n-1} b_{i,j}x^j$, where $\{b_{i,0}, b_{i,1}, b_{i,2}, ..., b_{i,n-1}\} \in Z_q^*$ are random numbers. Finally, the system public parameters are $pp = (G, Z_q^*, H_1, H_2, H_3, p, q, g, l, f_i(x)_{i\in[1,n]}, \{b_{i,c}\}_{c\in[0,n-1]})$.

- $\{U_i\}_{i\in[1,n]}$ convert their identity to $id : \{0,1\}^* \to Z_q^*$ and each U_i sends the message $msg_{ij} = ((f_i(id(U_j)), \{g^{f_i(id(U_k))}\}_{k\in[1,n]}, g^{b_{i,0}}))$ to U_j, where $g \in G$ is a random generator and $j \in \{1,2,3,...,n\}\backslash\{i\}$.

- U_j receives msg_{ij} from U_i, it checks whether $(g, f_i(id(U_j)), g^{f_i(id(U_j))})$ is correct and meanwhile $\prod_{m=1}^{n} g^{\zeta_m \cdot f_i(id(U_m))} = g^{b_{i,0}}$ holds, where $\zeta_m = \prod_{i=1,i\neq m}^{n} \frac{id(U_i)}{id(U_i)-id(U_m)}$. If the msg_{ij} holds, the identity of user nodes $\{U_i\}_{i\in[1,n]}$ can reach a consensus.

- U_i computes the private key $sk_i = \sum_{j=1}^{n} f_i(id(U_j))$, the public key $pk_i = g^{sk_i}$, the master private key $SK = \sum_{i=1}^{n} sk_i$, and the master public key $PK = g^{SK}$, where $i \in \{1,2,3,...,n\}$ and $j \in \{1,2,3,...,n\}\backslash\{j\}$.

5.3. The Process of Collaborative Deployment Smart Contracts

We deploy three smart contracts on three blockchains. SC deploys contract S_s, which processes and updates the data stored in SC and transfers the audit proof and data update requests. TC deploys contract S_t, which generates the audit proof and updates the data according to the requests of SC. AC deploys contract S_a, which audits the consistency of data in cross-chain interaction. S_s, S_t, and S_a have some correlations during the process of consistency verification. Hence, we deploy these smart contracts collaboratively. In the process of the deployment of contracts, we introduce Cosi signature protocol Syta et al. (2016) to ensure the security and privacy of cross-chain transfer requests and smart contracts. As shown in Fig. 4, the detailed procedures are as follows:

- The national regulatory authority deploys S_a on AC. Then S_a sends a deployment request to SC and TC, which needs to be satisfied by S_s and S_t.

- Upon receiving the request, the node on SC and TC starts to formulate the smart contract. Then, SC and TC sign the contract and send it to AC.

- AC verifies the correctness of the signature and checks whether the contract satisfies the request by running test data. If the contract sent by the corresponding blockchain meets the request, AC will sign the contract and return it to the corresponding blockchain for deployment. Otherwise, the process of

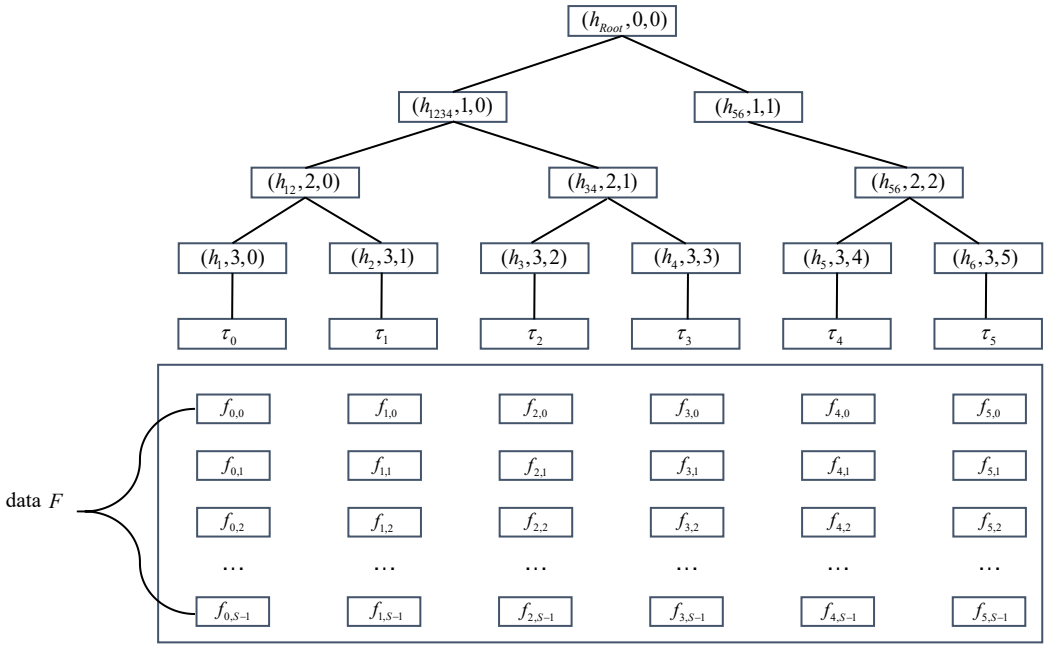

Fig. 3: The rule of constructing Merkle hash tree.

deployment will repeat until the contract meets the request.

In the process of collaboratively deploying smart contracts, the security of the transmitted requests and smart contracts must be guaranteed. Thus, we introduce Cosi signature protocol to sign the requirements and smart contracts. The specific procedures of signature and verification by Cosi protocol are as follows:

(1) The process of the signature by Cosi protocol

Requests and smart contracts will be signed by the leader U_{leader} selected from nodes $\{U_i\}_{i\in[1,n]}$ on their respective chains as follows:

- Each U_i computes $V_i = g^{v_i}$ and sends it to U_{leader}, where $v_i \in Z_q^*$ is selected randomly.
- U_{leader} computes $V = \prod_{i=1}^{n} V_i$ and $sig = H_1(V||msg)$, where msg is the signed message.
- U_{leader} broadcasts sig to $\{U_i\}_{i\in[1,n]}$.
- Each U_i returns $r_i = v_i - sig * sk_i$ to U_{leader}.
- U_{leader} computes $r = \sum_{i=1}^{n} r_i$ with the received $\{r_i\}_{i\in[1,n]}$.
- U_{leader} sends the signature (sig,r), msg, and V to the receiver.

(2) The process of verifying the signature

When AC receives the designed contracts, TC and SC receive the requests from AC, they verify the correctness of the signature as follows:

- With the received data (sig, r), msg, and V, verify the validity of (sig, r) with (1):

$$V = g^r \cdot PK^{sig} \tag{1}$$

If it holds, it means sig and r are honestly generated from the original message msg.

- Verify the security of msg with (2):

$$sig = H_1(V||msg) \tag{2}$$

If it holds, it means msg has not been maliciously tampered with or lost in the cross-chain transmission.

5.4. Data Transferring

Before storing data $F \in \{0,1\}^*$ to SC, DO conducts preliminary processing of F as follows:

- DO divides F into B blocks and each block is divided into S sectors $\{f_{ij}\}_{i\in[0,B-1],j\in[0,S-1]} \in Z_q^*$, where each block has the same size of $S|q| = \lceil |F|/B \rceil$. Meanwhile, if the last block is not the same size as the other blocks, DO will pad 0 at the end of the last block.
- DO selects a random value $r_d \in \mathbb{Z}_q^*$ and calculates $R_d = g^{r_d}$. Then, DO blinds F with $\{f'_{ij}\}_{i\in[1,B],j\in[1,S]} = \{f_{ij}\}_{i\in[1,B],j\in[1,S]} + H_2(fd||r_d)$ to protect its security and privacy, where $fd \in \{0,1\}^*$ is the data identifier.
- DO computes the tag $\tau_i = \prod_{j=0}^{S-1} g^{f'_{ij}}$ and the random value of decentralized chameleon hash $r_0 = (g^e, PK^e)$, where $i \in [0, B-1]$, $j \in [0, S-1]$, and $e \in Z_q^*$ is a random number.

- DO uploads $\{\{f_{ij}^{'}\}_{i\in[0,B-1],j\in[0,S-1]}, \{\tau_i\}_{i\in[0,B-1]}, r_0\}$ to SC for further processing.

- SC constructs a Merkle hash tree (MHT) to store $\{f_{ij}^{'}\}_{i\in[0,B-1],j\in[0,S-1]}$ and each node $node$ in MHT stores the tuple $addr = \{h_{node}, l_{node}, r_{node}\}$, where h_{node} represents the hash value, $l_{node} \in [0, \lceil log_2^B \rceil]$ denotes the number of layers from top to bottom, and $r_{node} \in [0, B-1]$ indicates the number of its position sequence from left to right. We introduce the decentralized chameleon hash function here to modify the original MHT. If $node$ is a leaf node, compute $u = H_1(m, PK)$, $h_{node} = g^e u^m$, where $m = (r_0||\tau_0||...||\tau_{S-1}||l_{node}||r_{node})$. Otherwise, $u = H_1(m, PK)$, $h_{node} = g^e u^m$, where $m = (h_l||h_r||l_{node}||r_{node})$, h_l and h_r denote the hash values stored by the left and right child nodes. Finally, the root hash h_{Root} of MHT can be achieved and SC generates a data uploading transaction Tx_1 and sends it to TC, which includes $\{\{f_{ij}^{'}\}_{i\in[0,B-1],j\in[0,S-1]}\}, \{\tau_i\}_{i\in[0,B-1]}\}$.

In order to ensure the security and privacy of the transferred transaction, we adopt multi-signcryption algorithm to process the transferred transaction. The specific procedures are as follows:

- U_i in SC selects random number $r_i \in Z_q^*$ and $\kappa_i \in Z_q^*$. Then, U_i calculates $\lambda_i = g^{\kappa_i}$, $T_i = \lambda_i + sk_i \cdot H_2(id(U_i), \lambda_i, pk_i)$ and achieves the timestamp (λ_i, T_i) of the signed message $F^{'}$ while the recipient U_{TC} in TC also obtains (λ_{TC}, T_{TC}) and discloses T_{TC}, where $H_2 : \{0,1\}^* \times Z_q^* \rightarrow Z_{q-1}^*$. Besides, U_i calculates $\sigma_i = g^{r_i}$ and broadcasts σ_i to other signers.

- U_{leader} calculates $\sigma = \prod_{i=1}^{n} \sigma_i$.

- U_i computes $h = H_2(msg, \sigma)$, where msg is the signed message $F^{'}$.

- U_i computes $A_i = msg \oplus H_3(pk_{TC}^{T_{TC}} \cdot g^{sk_i \cdot T_i})$, where pk_{TC} is the public key of U_{TC}, sk_i is the private key of U_i, and $H_3 : Z_q^* \rightarrow \{0,1\}^l$.

- U_i computes $\eta_{TC} = H_3(msg, T_{TC})$.

- U_i computes $\beta_i = sk_i(\eta_{TC} + T_{TC} + h \cdot \sigma) - r_i$.

- U_i outputs $(\sigma_i, \sigma, \beta_i, h, T_i, T_{TC}, A_i)$ and checks whether $pk_i^{\eta_{TC}+T_{TC}+h\cdot\sigma} = \sigma_i g^{\beta_i}$ holds. If the equation holds, computes $\beta = \sum_{i=1}^{n} \beta_i$.

- Output the final signed ciphertext $F^{'*} = (\sigma, \beta, h, T_i, T_{TC}, \{A_i\}_{i\in[1,n]})$ to TC.

Upon receiving the Tx_1 from SC, TC firstly checks the signcryption of $F^{'}$ as follows:

- Compute $msg = A_i \oplus H_3(g^{sk_{TC}\cdot T_{TC}} \cdot pk_i^{T_i})$, where sk_{TC} is the private key of U_{TC} and pk_i is the public key of U_i.

- Compute $\eta_{TC} = H_3(msg, T_{TC})$.

- Verify whether $PK^{h\cdot\sigma+\eta_{TC}+T_{TC}} = \sigma g^{\beta}$ holds. If the equation holds, it means the transferred data $F^{'}$ is valid.

- Verify whether the uploaded data set $\{f_{ij}^{'}\}_{i\in[0,B-1],j\in[0,S-1]}$ is consistent with the tag set $\{\tau_i\}_{i\in[0,B-1]}\}$. If not, TC rejects to store $F^{'}$; otherwise TC stores the transferred data and generates a decentralized chameleon hash random value r_1.

5.5. Data Updating

Assuming that there are n user nodes $\{U_1, U_2, ..., U_n\}$ in SC. We sort these user nodes by the hash values of their identity. Then, we choose the first t user nodes to generate a secret key, which will be shared and utilized to update the transaction. Meanwhile, in order to prevent an attack compromising $t-1$ user nodes before and in the update phase, we implement the threshold to $2t-1 (2t-1 > (t-1)+(t-1))$ for updating. Besides, in order to support threshold updating among n user nodes, the secret key will be re-shared among them. Thus, only $t^{'}(t^{'} > t)$ user nodes can update the transaction collaboratively. The specific steps to update the transactions are described as follows:

- User nodes $\{U_i\}_{i\in[1,n]}$ generate the decentralized chameleon hash key, which utilize $DCH.KeyGen(1^l, 1^t) \rightarrow \{PK, sk_i\}$, where $t \leq \frac{n}{2}$.

- Each U_i chooses a $(2t-1)$-degree polynomial function $f_i(id(U_i), x) = sk_i + \sum_{j=1}^{2t-1} b_{i,j}x^j$, where $i \in \{1, 2, ..., t\}$ and $b_{i,1}, b_{i,2}, ..., b_{i,2t-1} \in Z_q^*$. Next, U_i sends the message $msg_{ij}^{'} = (f_i(id(U_i), id(U_j)), \{g^{f_i(id(U_i),id(U_k))}\}_{k=1}^{2t}, pk_i))$ to U_j, where $i \in \{1, 2, ..., t\}$ and $j \in \{1, 2, ..., 2t\} \backslash \{i\}$.

- Upon receiving $msg_{ij}^{'}$ from U_i, U_j firstly verifies whether $(g, f_i(id(U_i), id(U_j)), g^{f_i(id(U_i),id(U_j))})$ is correct and meanwhile $\prod_{m=1}^{n} g^{\zeta_m \cdot f_i(id(U_i),id(U_m))} = g^{b_{j,0}}$ holds, where $\zeta_m = \prod_{i=1,i\neq m}^{n} \frac{id(U_i)}{id(U_i)-id(U_m)}$. If the correctness verifies successfully, the user node U_i outputs a $(t-1)$-degree polynomial function $f(x, id(U_i))$ as well as t shared data $\{f_k(id(U_k), id(U_i))\}_{k=1}^{t}$, where $i \in \{1, 2, ..., 2t\}$. Finally, U_i sends the message $msg_{ij}^{''} = (f(id(U_j), id(U_i)), \{g^{f(id(U_k),id(U_i))}\}_{k=1}^{n}, g^{(0,id(U_i))})$ to U_j, where $i \in \{1, 2, ..., 2t\}$ and $j \in \{1, 2, ..., n\} \backslash \{i\}$. After receiving $msg_{ij}^{''}$ from U_i, U_j firstly checks whether the shared data holds in a similar way as

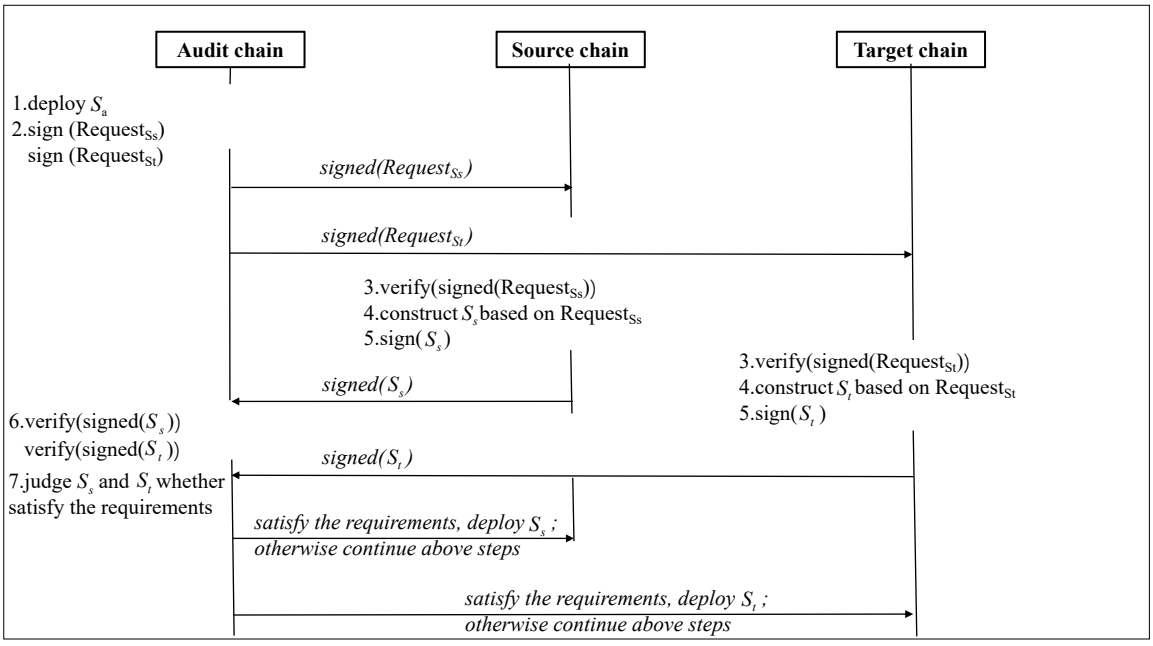

Fig. 4: The process of deploying the smart contract collaboratively.

above. If the check holds, the user node U_i outputs a $(2t-1)$-degree polynomial function $f(id(U_i), z)$ as well as $2t$ shared data $\{f_k(id(U_i), id(U_k))\}_{k=1}^{2t}$, where $i \in \{1, 2, ..., n\}$.

Due to the fact that 1) the user nodes in the blockchain network may join or leave anytime; 2) an adversary may collect more than t shared data to reset the private key, which is utilized to update the block, we introduce a method to adjust the threshold from (t, n) to $(t^{'}, n^{'})(t \leq t^{'}, t^{'} \leq \frac{n^{'}}{2})$, where $t^{'}$ is the new number of threshold and $n^{'}$ is the number of user nodes. The specific steps are as follows:

- U_i output a $(2t^{'}-1)$-degree polynomial function $f^{'}(x, id(U_i)) = f(x, id(U_i)) + F(x, id(U_i))$, where $F(x, y)$ is a random polynomial function of degree $(t^{'}-1, 2t^{'}-1)$, $F(0,0) = 0$, and $i \in \{1, 2, ..., 2t^{'}\}$.
- Update the private key shares from $f(x, y)$ to $f^{'}(x, y) = f(x, y) + F(x, y)$ of degree $(t^{'}-1, 2t^{'}-1)$ and $f^{'}(0,0) = f(0,0) = 0$.
- U_i generates new transactions $tx^{'} = \{tx_1^{'}, tx_2^{'}, ..., tx_\rho^{'}\}$ to update the original transactions $tx = \{tx_1, tx_2, ..., tx_\rho\}$. The address of tx are $addr = \{addr_{tx_1}, ..., addr_{tx_\rho}\}$ and ρ is the number of the updated transactions.
- U_i verifies the signature of $(tx_i, tx_i^{'})(i \in \{1, 2, ..., \rho\})$ to ensure tx_i is generated by the same user node. If the signature is correct, U_i replaces the address of tx by $tx^{'}$ and generates a new Merkle hash root h_{Root}^{*}.
- U_i generates the data update request $req = (addr_{tx}, \{tx, tx^{'}\}, h_{Root}^{*})$ and broadcasts the request req to the other user nodes $\{U_1, U_2, ..., U_n\}\backslash\{U_i\}$ in the entire blockchain network of SC. Upon receiving the request req, they verify the transaction pairs $(tx_i, tx_i^{'})(i \in 1, 2, ..., \rho)$ and if the user nodes $\{U_1, U_2, ..., U_t\}$ agree with the request req, they will send the confirm information $(id(U_1), id(U_2), ..., id(U_t))$ to U_i, where $t^{'} > t$. Last, the new randomness of tx will be computed $r_0^{'} = (g^{e^{'}}, PK^{e^{'}})$, where $e^{'} = g^e \cdot u^{(tx - tx^{'})}$ and $u = H_1(PK, tx)$. Due to the feature of chameleon hash function, we can update the tx without altering the hash of the block header.
- U_i generates the hashes $hash_{tx^{'}} = (H(tx_1^{'}), ..., H(tx_\rho^{'}))$ and packages the data update request $req^{'} = (addr_{tx}, hash_{tx^{'}})$. Then, U_i utilizes multi-signcryption algorithm to sign the $req^{'}$, which is similar to the steps in Section 5.4.
- SC sends data update request $req^{'}$ to TC. After receiving the data update request $req^{'}$, the user node U_j in TC firstly checks the correctness of the signature. If the signature is correct, the user node U_j in TC locates the address of the transaction tx, then updates tx in a similar way above in SC to generate a new randomness $r_1^{'}$ and h_{Root}^{**}.

VerifyConsistency. To ensure consistency in cross-chain interaction, AC will send challenge $chal$ to SC and TC to audit the consistency of data. The audit of AC can be divided into two aspects.

- The consistency of transmitted data in the cross-chain interaction between SC and TC.

- The consistency of dynamic update of data between SC and TC.

We will give a concrete depiction of the auditing process of AC.

(1) The process of data consistency auditing periodically

AC periodically executes a consistency audit to check the consistency of transmitted data in the cross-chain interaction between SC and TC. It invokes contract S_a to generate a challenge $chal$ and sends it to SC and TC. After receiving the proof $proof_{SC}$ and $proof_{TC}$ from SC and TC respectively, it invokes contract S_a to verify the integrity of $F^{'}$. The specific procedures of periodic audit are as follows:

- AC generates a random challenge $chal = \{addr_i\}_{i=0}^{M-1}$, which contains $M \leq S$ sectors to be challenged and $addr_i$ ($i \in [0, M-1]$) is the position of challenged sectors.
- AC sends $chal$ to SC and TC.
- Upon receiving $chal$, SC generates proof $proof_{SC} = (\{\tau_{addr_i}\}_{i=0}^{M-1}, \{\tau_i^*\}_{i=0}^{M-1}, h_{Root})$, where $\{\tau_{addr_i}\}_{i=0}^{M-1}$ are the tags of challenged sectors, $\{\tau_i^*\}_{i=0}^{M-1}$ are the siblings of the challenged sectors from the leaf node to the root node, and h_{Root} is the root of MHT.
- Upon receiving $chal$, TC generates proof $proof_{TC} = (\{\tau^{'}_{addr_i}\}_{i=0}^{M-1}, \{\tau_i^{*'}\}_{i=0}^{M-1}, h^{'}_{Root})$ in a similar way as SC.
- After receiving the proof $proof_{SC}$ and $proof_{TC}$, AC invokes S_a to compute $h_{addr_i} = H(\tau_{addr_i})$ and $h^{'}_{addr_i} = H(\tau^{'}_{addr_i})$, where $i \in \{0, 1, ..., M-1\}$ and compares whether the newly generated Merkle hash root is equal to h_{Root}. If all the verification passes, it means the transmitted data are consistent with the data on SC.

(2) The process of data updating consistency audit

After updating data in SC, SC will send an update request to AC to notify AC to conduct a data update consistency audit. Then, AC will send a $chal$ to SC and TC. Upon receiving $chal$, SC and TC generate proofs for auditing. The details of the audit are as follows:

- Upon receiving $chal$ from AC, SC generates $proof_{SC} = \{pk_{U_{SC}}, \{tx_i, tx_i^{'}\}_{i=0}^{\rho}\}, \{r_0, r_0^{'}\}\}$, where $pk_{U_{SC}}$ is the data updating initiator in SC, $\{tx_i, tx_i^{'}\}_{i=0}^{\rho}$ and $\{r_0, r_0^{'}\}$ are the transaction information and decentralized chameleon hash randomness number before and after updating in SC. TC generates $proof_{TC} = \{pk_{U_{TC}}, \{r_1, r_1^{'}\}\}$, where $pk_{U_{TC}}$ is the user node to update $F^{'}$ in TC and $\{r_1, r_1^{'}\}$ is the decentralized chameleon hash randomness number before and after updating in TC.
- After receiving $proof_{SC}$ and $proof_{TC}$ from SC and TC respectively, AC invokes S_a to verify the correctness of $DCH.Verify(pk_{U_{SC}}, (tx_i, r_0), (tx_i^{'}, r_0^{'}))$ and $DCH.Verify(pk_{U_{TC}}, (tx_i, r_1), (tx_i^{'}, r_1^{'}))$, where $i \in [0, \rho]$. If they all output 1, it indicates the data update was carried out as required. Otherwise, data consistency verification fails.

6. Security Analysis

In this section, we conduct a theoretical security analysis of dynamic cross-chain decentralized data consistency verification model (DCCV), including the security analysis of cross-chain tampering attacks, cross-chain privacy leakage attacks, and audit inconsistency attacks that exist in the process of cross-chain interaction.

Cross-chain tampering attacks: In the process of cross-chain interaction, if the transferred data $F^{'}$ or update request is intercepted by an adversary, it may cause a tampering attack.

Theorem 1. *The privacy of data in cross-chain interaction is preserved if and only if the Discrete Logarithm Problem (DLP) holds in multi-signcryption algorithm.*

Proof: If an adversary wants to forge or tamper with the transferred data, it needs to forge the private key of at least one signer. With the public key PK, it is hard to find the corresponding private key according to the Discrete Logarithm Problem Odlyzko (2000). Thus, the forged multi-signcryption message cannot meet $PK^{h\sigma+\eta_{TC}+T_{TC}} = \sigma g^{\beta}$.

Theorem 2. *The security of the transferred contracts and requests among chains realizes if the sender can generate a correct proof of the transferred message and the smart contract installed on the receiver can always pass (3).*

Proof: By simplifying (3), it can be seen that the receiver could always make a correct judgment about the security of the transferred contracts and requests among chains. The details are as follows:

$$
\begin{aligned}
V^{'} &= g^{r} PK^{sig} = g^{\sum_{i=1}^{n} r_i} PK^{sig} \\
&= g^{\sum_{i=1}^{n} v_i - sig \cdot sk_i} PK^{sig} = g^{\sum_{i=1}^{n} v_i - \sum_{i=1}^{n} sig \cdot sk_i} PK^{sig} \\
&= \frac{g^{\sum_{i=1}^{n} v_i}}{g^{\sum_{i=1}^{n} sig \cdot sk_i}} PK^{sig} = \frac{(g^{v_1} g^{v_2} g^{v_3}, ..., g^{v_n})}{(g^{sk_1} g^{sk_2} g^{sk_3}, ..., g^{sk_n})^{sig}} PK^{sig} \\
&= \frac{\prod_{i=1}^{n} V_i}{(\prod_{i=1}^{n} PK_i)^{sig}} PK^{sig} = \frac{V}{PK^{sig}} PK^{sig} = V
\end{aligned}
\tag{3}
$$

Cross-chain privacy leakage attacks: In the process of cross-chain interaction, if the plaintext content of cross-chain transferred data is achieved by an adversary, it may cause a privacy leakage attack.

Theorem 3. If an adversary wants to achieve the plaintext content of cross-chain transferred data if and only if DLP holds in multi-signcryption algorithm.

Proof: If an adversary wants to achieve the plaintext content of cross-chain transferred data, it needs to forge the signature without obtaining the private key of the receiver in TC. It is hard to achieve the purpose according to DLP. Thus, $C_i = g^{sk_{TC} \cdot T_{TC} pk_i^{T_i}}$ cannot be calculated and the plaintext $msg = A_i \oplus H_3(C_i)$ also cannot be obtained.

Audit inconsistency attacks: In the process of cross-chain data consistency auditing, if TC can realize the purpose of forging a correct $proof$ of the transferred data to pass the consistency auditing of AC, it may cause an audit inconsistency attack.

Theorem 4. If TC can forge a correct $proof$ of the transferred data to pass the consistency auditing of AC if and only if the hash collisions and DLP hold.

Proof: For the one hand, TC needs to forge a h_{Root} to pass the periodical consistency auditing. It is hard to achieve the purpose according to the hash collisions. For the other hand, TC needs to generate a randomness number r_1 to pass the verification of the decentralized chameleon hash function. It is difficult to implement according to DLP.

7. Experimental Evaluation

In this section, we conduct experiments to evaluate the performance of DCCV in terms of practicality and efficiency. We deploy DCCV on Hyperledger Fabric v1.4 on Ubuntu18.04 with 32GB of memory and simulate the entire experimental process by running the three different smart contracts S_a, S_s, and S_t based on the Go language.

7.1. Data Preprocessing Overhead

In DCCV, data need to be preprocessed by DO and then stored in SC. We compare the preprocessing time for different block sizes and the results are shown in Fig. 5. As can be seen from the figure, with the increase of block size, the data preprocess time decreases. The increase of block size reduces the number of divided blocks, so the data preprocess time also decreases. Besides, we conduct the experiment to compare the data preprocessing overhead for different sector numbers in a block, which is shown in Fig. 6. As the sectors in a block increase, DO needs to calculate more tags to build the Merkle hash tree, resulting in the time overhead accordingly.

7.2. Data Updating and Verifying Overhead

We design experiments to verify the decentralized chameleon hash function for updating and verifying time compared with different numbers of thresholds, which is shown Fig. 7. As we can see from Fig. 7, the verification time overhead remains stable. The verification time overhead mainly includes verification of signatures and consistency and the time complexity of these transactions is relatively stable. However, as the number of different thresholds increases, it takes more time for user nodes to reach a consensus in updating.

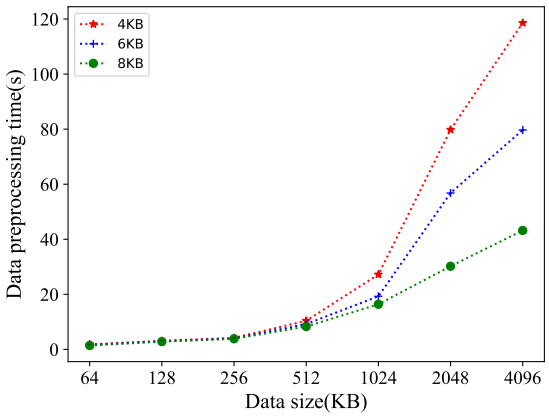

Fig. 5: The time overhead of data preprocessing with the data size increases.

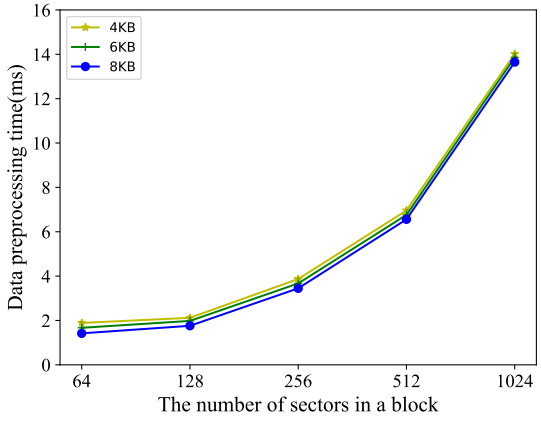

Fig. 6: The time overhead of data preprocessing with the sectors in a block increases.

7.3. Hash Key Generation and Hash Computation

The time overhead variation of key generation and hash calculation of the decentralized chameleon hash function is designed under different numbers of thresholds. The results are shown in Fig. 8. The time overhead of Hash and ReHash in the decentralized chameleon hash function is basically stable and the key generation time increases with the increase of threshold number. The increase of threshold number is accompanied by the increase of the number of user nodes, which share a decentralized chameleon hash key in the blockchain. Thus, time overhead increases accordingly.

7.4. Multi-signcryption Algorithm Overhead Comparison

In DCCV, we introduce multi-signcryption algorithm to ensure the security and privacy of the transferred data in cross-chain interaction. We compare multi-signcryption algorithm in this paper with representative multi-signature algorithm schemes Jalil et al. (2022); Cao and Cao (2009); Shao (2009); Du and Wen (2014) and the results are shown in

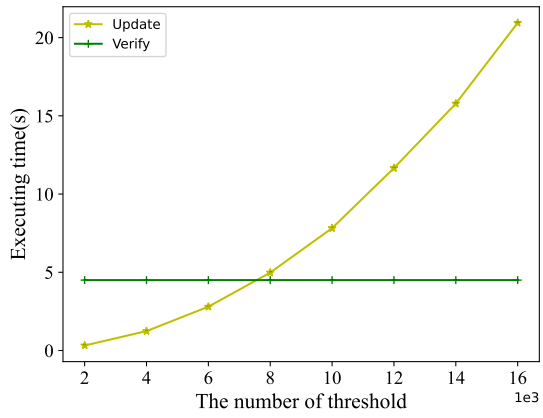

Fig. 7: The time overhead of decentralized chameleon hash function updating and verifying.

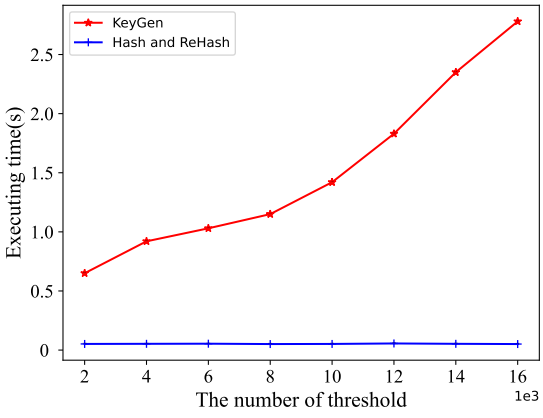

Fig. 8: The time overhead of key generation and hash.

Table 2
Multi-signcryption algorithm overhead comparison

Reference	Sign/ms	Verify/ms	Total/ms
RefJalil et al. (2022)	1.6375	3.8243	5.4618
RefCao and Cao (2009)	2.4625	1.3674	3.8299
RefShao (2009)	2.2645	1.4286	3.6391
RefDu and Wen (2014)	1.3176	3.2183	4.5359
Our scheme	1.3026	1.0321	2.3347

Table 2. As we can see from Table 2, the efficiency of multi-signcryption algorithm introduced in this paper is higher than that in Jalil et al. (2022); Cao and Cao (2009); Shao (2009); Du and Wen (2014). It can achieve the purpose of ensuring the security and privacy of the transferred data as well as having high efficiency in cross-chain interaction.

7.5. Overhead of Cross-chain Data Transferring

We conduct the experiment to evaluate the overhead of cross-chain data transferring with the increasing of data size and the result is shown in Fig. 9. As the data size transmitted in cross-chain interaction increases, the overhead of data transmission among different chains also increases.

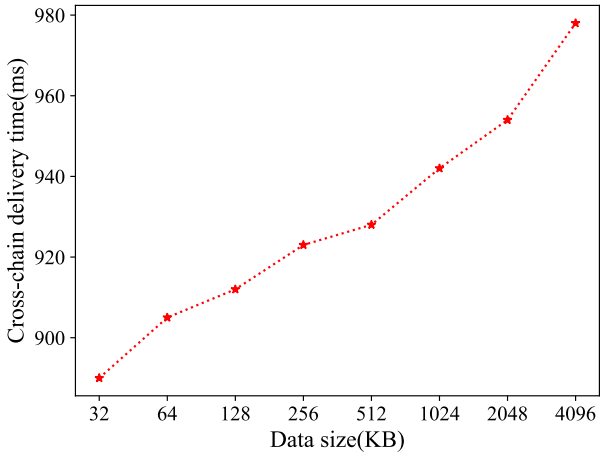

Fig. 9: The overhead of data transfer among chains as the data size increases.

7.6. Cosi Protocol Execution Time Evaluation

we conduct experiments to assess the efficiency of the Cosi protocol, in comparison to other commonly used multi-signature schemes such as RSA-based and BLS-based multi-signature. The results of the experiment are presented in Fig. 10. It is important to note that although the execution time of the RSA-based multi-signature during the verification process is the lowest, the overall time overhead is similar to that of Cosi, which indicates the efficiency of Cosi. Besides, we consider the variation of execution time with the number of signers, which is shown in Fig. 11, which indicates the scalability and efficiency of Cosi.

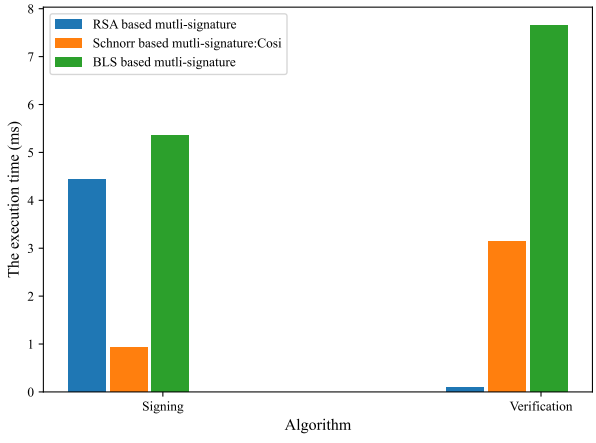

Fig. 10: The execution time of typical multi-signature schemes.

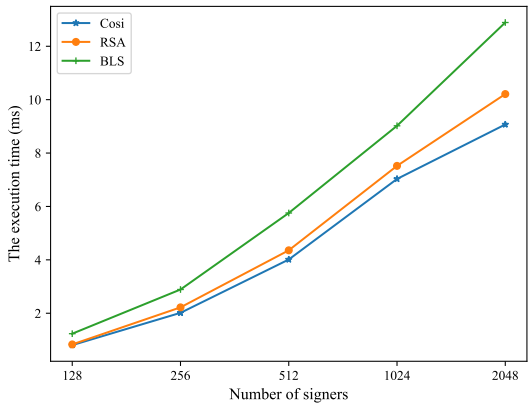

Fig. 11: The execution time of Cosi, RSA, and BLS in different numbers of signers.

8. Conclusion

In this paper, we design a decentralized consistency verification model (DCCV) for dynamic data in cross-chain interaction to guarantee the consistency of dynamic data. We adopt a decentralized chameleon hash function to update the transaction. In order to make the consistency verification more frequent and improve efficiency to develop smart contracts, we implement a collaborative way to develop smart contracts. Besides, we utilize Cosi protocol and multi-signcryption to ensure the security and privacy of the transferred information. Finally, we conduct security analysis and experimental evaluation to demonstrate that DCCV is secure and efficient in practice. We envision to deploy DCCV on a public blockchain, e.g., Ethereum to assess its efficiency and availability in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

O. I. Abiodun, M. Alawida, A. E. Omolara, and A. Alabdulatif. Data provenance for cloud forensic investigations, security, challenges, solutions and future perspectives: A survey. *J. King Saud Univ. Sci.*, 2022.

G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik. Scalable and efficient provable data possession. In *Proc. 4th Int. Conf. Secur. Privacy Commun. Netw.*, pages 1–10, 2008.

N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts. In *Proc. 6th Int. Conf. Principles Secur. Trust*, pages 164–186. Springer, 2017.

A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. Enabling blockchain innovations with pegged sidechains. *Sidechains protocol white paper*, 72:201–224, 2014.

F. Cao and Z. Cao. A secure identity-based proxy multi-signature scheme. *Information Sci.*, 179(3):292–302, 2009.

X. Deng, B. Li, S. Zhang, and L. Deng. Blockchain-based dynamic trust access control game mechanism. *J. King Saud Univ. Sci.*, 2023.

H. Du and Q. Wen. Certificateless proxy multi-signature. *Information Sci.*, 276:21–30, 2014.

C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. *ACM Trans. Inf. Syst. Secur.*, 17(4):1–29, 2015.

A. G. Gad, D. T. Mosa, L. Abualigah, and A. A. Abohany. Emerging trends in blockchain technology and applications: A review and outlook. *J. King Saud Univ. Sci.*, 2022.

Z. Haddad. Blockchain-enabled anonymous mutual authentication and location privacy-preserving scheme for 5g networks. *J. King Saud Univ. Sci.*, 2022.

K. He, J. Chen, Q. Yuan, S. Ji, D. He, and R. Du. Dynamic group-oriented provable data possession in the cloud. *IEEE Trans. Depend. Sec. Comput.*, 18(3):1394–1408, 2019.

A. Hope-Bailie and S. Thomas. Interledger: Creating a standard for payments. In *Proc. Int. Conf. Companion World Wide Web*, pages 281–282, 2016.

B. A. Jalil, T. M. Hasan, G. S. Mahmood, and H. N. Abed. A secure and efficient public auditing system of cloud storage based on bls signature and automatic blocker protocol. *J. King Saud Univ. Sci.*, 34(7):4008–4021, 2022.

M. Jia, J. Chen, K. He, R. Du, L. Zheng, M. Lai, D. Wang, and F. Liu. Redactable blockchain from decentralized chameleon hash functions. *IEEE Trans. Inf. Forensics Security*, 17:2771–2783, 2022.

J. Jiang, Y. Zhang, Y. Zhu, X. Dong, L. Wang, and Y. Xiang. Dciv: Decentralized cross-chain data integrity verification with blockchain. *J. King Saud Univ. Sci.*, 34:7988–7999, 2022.

D. Li, J. Liu, Z. Tang, Q. Wu, and Z. Guan. Agentchain: A decentralized cross-chain exchange system. In *Proc. 18th IEEE Int. Conf. Trust, Security and Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. and Eng.*, pages 491–498. IEEE, 2019.

A. Odlyzko. Discrete logarithms: The past and the future. In *Des. Codes Cryptography*, pages 59–75. Springer, 2000.

J. Patil and S. Chaudhari. Privacy preserving and dynamic audit service for secure cloud storage. In *Proc. Int. Conf. Smart City Emerg. Technol.*, pages 1–6. IEEE, 2018.

J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

Z. Shao. Improvement of identity-based proxy multi-signature scheme. *J. Syst. Software*, 82(5):794–800, 2009.

S. Sharmila Deva Selvi, S. Sree Vivek, R. Srinivasan, and C. Pandu Rangan. An efficient identity-based signcryption scheme for multiple receivers. In *Proc. Int. Workshop on Secur.*, pages 71–88. Springer, 2009.

J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo. An efficient public auditing protocol with novel dynamic structure for cloud data. *IEEE Trans. Inf. Forensics Security*, 12(10):2402–2415, 2017.

B. Shrimali and H. B. Patel. Blockchain state-of-the-art: architecture, use cases, consensus, challenges and opportunities. *J. King Saud Univ. Sci.*, 34(9):6793–6807, 2022.

E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *IEEE Symp. Security Privacy*, pages 526–545, 2016.

W.-T. Tsai, R. Blower, Y. Zhu, and L. Yu. A system view of financial blockchains. In *Proc. IEEE Symp. Service-Oriented Syst. Eng.*, pages 450–457. IEEE, 2016.

C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou. Toward secure and dependable storage services in cloud computing. *IEEE Trans. Services Comput.*, 5(2):220–232, 2012. doi: 10.1109/TSC.2011.24.

Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Trans. Parallel Distrib. Syst.*, 22(5):847–859, 2010.

J. Xu, A. Yang, J. Zhou, and D. S. Wong. Lightweight delegatable proofs of storage. In *Proc. 21st Eur. Symp. Res. Comput. Security*, pages 324–343. Springer, 2016.

A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong. Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage. *IEEE Trans. on Cloud Comput.*, 9(1):212–225, 2018.

Y. Zhang, J. Jiang, X. Dong, L. Wang, and Y. Xiang. Bedcv: Blockchain-enabled decentralized consistency verification for cross-chain calculation. *IEEE Trans. Cloud Comput.*, pages 1–12, 2022.