Secure and Efficient Data Interoperability Protocols for Multi-Blockchains Systems

Teng Yu, Fengji Luo, *Senior Member*, *IEEE*, Gianluca Ranzi, *Member IEEE*, and Jianzhong Wu, *Fellow*, *IEEE*

Abstract [1] **—The proliferation of decentralized applications across different autonomous blockchains raises the need to enable cross-chain data interoperability (CCDI). However, prior approaches for supporting CCDI often hit scalability bottlenecks regarding critical metrics, e.g., memory, or remain prone to withholding and censorship attacks. This paper proposes two protocols to implement secure and efficient CCDI under adversarial conditions. The cross-chain token exchange (CCTE) protocol for atomic swaps is proposed. It adopts a deposit mechanism, a blockchain-of-blockchains (BoB), and Merkle proofs to ensure the completion of token exchanges even under withholding attacks. It utilizes a parallelized design to support concurrent token exchanges, thereby improving its efficiency and avoiding censorship attacks that target sequential token exchanges. The CCDI protocol is proposed to support any CCDI application. It authorizes a unique BoB to execute arbitrary CCDI application logic. It integrates a "transfer and in place data update" mechanism to improve its efficiency, and this mechanism enables a blockchain update its state data items using a single transaction, without requiring any information from other blockchains. Moreover, the CCDI protocol integrates a state data migration scheme, which supports a user to migrate its state data item to censorship-resilient blockchains, and incorporates a malicious user nodes elimination scheme, which enables the updates of state data items in a CCDI process even under withholding attacks. Systematic performance evaluations are conducted to compare the two protocols with existing ones. The CCTE protocol reduces latency by at least 52% compared to existing protocols under probabilistic consensus setting. The CCDI protocol outperforms prior protocols, lowering communication cost by 59%, computation overhead by 41%, memory burden by 12%, and latency cost by 33%.**

Index Terms— **blockchain, cross-chain data interoperability, cross-chain token exchange, atomic swaps**

I. INTRODUCTION

SINCE from its proposal in 2008, blockchain (BC) [1] has been extensively practiced in different fields. The wide applications of BC raise the need of Cross-Chain Data Interoperability (CCDI), referring to as a data integration paradigm that the data stored in one BC is reachable, verifiable, and referable by another in a semantically compatible manner [2]. A typical CCDI application is the Cross-Chain Token Exchange (CCTE) (or atomic swaps) [3-10], which allows a party to transfer its tokens to another party in a BC if and only if the latter transfers its tokens to the former in another BC. Besides facilitating CCTE application, CCDI also supports more complicated data interactions across autonomous BCs [11-30] – for example, for social media and Esports applications deployed in different BCs, CCDI can not only enable users in a same BC to interact but also support a user to interact the users from other BCs.

One challenge of CCDI lies in the inherent independence of BCs [1], which restricts that the nodes in a BC can only access the state data (SD) of the BC (e.g., account balances and contracts statuses) and cannot access the SD of other BCs. Such an independence nature of BCs would hinder CCDI operations. Most approaches directly compromise BCs' independence property to support CCDI. However, lacking BCs' independence may introduce additional issues. For instance, a BC must retain extra information from each connected BC to validate cross-chain data, causing significant memory overhead as the number of connected BCs grows.

Another challenge of CCDI is to guarantee the atomicity property [2] in the presence of withholding and censorship attacks. Specifically, the atomicity property states a situation where all SD items involved in a CCDI process is either fully updated or completely unchanged [2]. Under withholding attacks, a malicious user can reject to submit its transaction, which stalls the updates of all SD items involved in a CCDI process, thereby denying CCDI services [5]. More seriously, under censorship attacks, the atomicity property may not hold anymore when a BC censors a transaction required in a CCDI process, which directly threats CCDI security [1].

A. State-of-the-Art

A number of protocols have been developed to address the above two challenges in a multi-blockchain system (MBS).

The CCTE scenario. Time-lock-based mechanisms are one of the solutions to enable the independence of BCs and support CCTE. The "lock" operation can be implemented in different ways, such as hash-time locks [5, 6, 8], signature-based locks [7, 10] and time-lock puzzles [9]. The original hash time lock contract (HTLC) scheme [5] requires a larger time cost for the "lock" operations to ensure atomicity of a CCTE process, which is time efficient. Recent work further improves the time efficiency of these mechanisms by moving the locking phase off-chain and employing adaptor signatures to carry out the

[1]This work is supported by Australian Research Council through a Discovery Project (DP220103881).

T. Yu, F. Luo and G. Ranzi are with Faculty of Engineering, The University of Sydney, NSW 2006, Australia (emails: teyu7403@uni.sydney.edu.au; fengji.luo@sydney.edu.au; gianluca.ranzi@sydney.edu.au).

J. Wu is with School of Engineering, Cardiff University, United Kingdom (email: WuJ5@cardiff.ac.uk).

unlocking phase on-chain [7, 10].

Despite these merits, time-lock-based mechanisms remain vulnerable to withholding and censorship attacks. By design, these mechanisms mandate that one party must complete its token transfer before its counterparty initiates theirs [5-10]. If the party is malicious, it can simply withhold that transfer, stalling the entire conditional exchange and wasting its counterparty's time. Even if the party's token transfer appears on its BC, the counterparty's transfer on the target BC can be censored for a certain time window. During this time window, the party may initiate another transaction to spend out its tokens, causing the counterparty's token transfer to fail after the time window. Although fungibility property[2] is a promising solution to counter censorship attacks [7], it breaks down entirely if the party colludes with the target BC.

Recent work [3 ,4] proposes a CCTE protocol that preserves the BCs' independence while ensuring resilience to censorship attacks. The protocol in [3] sets up relay nodes to enable CCTE and relies on a blockchain of blockchains (BoB) to ensure the integrity and trustworthy of the relay nodes. [4] extends the CCTE framework to support multi-party scenarios. However, the work in [3] has two significant limitations: firstly, its mechanistic design is not efficient, imposing higher communication, computation, and latency overhead than time-lock-based schemes; secondly, under withholding attacks, the protocol in [3] must roll back all CCTE operations to ensure its atomicity property, thereby denying CCTE services.

Based on the above literature review and discussion, we ask the first research question: *whether there exists a protocol that can (i) ensure the atomicity property under censorship attacks, (ii) progress CCTE services even in the presence of withholding attacks, and (iii) offer substantially higher efficiency, regarding the metrics of communication, computation, memory and latency, than the CCTE protocol in [3]?*

The CCDI scenario. There are two paradigms of CCDI protocols to support complex CCDI applications, e.g., social media and Esports. In the first paradigm, a BoB is introduced in an MBS, but its role is simplified to just forward or translate a CCDI request from the source BC to the target BC [12-20]. On this basis, the CCDI protocols facilitate direct state data verification between BCs, which thus require each BC to store the block header of the others – this in turn compromises the independence property of involved BCs.

The second paradigm of CCDI protocols also rely on a BoB, but it assigns the BoB with enhanced abilities [21-30]. Specifically, the BoB in [24-26] can create SD items from its connected BCs and execute any CCDI logic. The BoB can also issue a statement (also known as a certificate [24-27] or a proof [28-30]) for a CCDI request and send it to relevant BCs. However, to verify the BoB's statements, the BC needs to cache the extra information (e.g., block hashes of the BoB), which compromises the independence property of these BCs, thereby burdening them with extra memory cost.

Notably, the protocols of the two paradigms [12-30] execute the operations for the updates of SD items within a single BC – either a BoB or a target BC. With this design, the protocols can be censorship-resilient through selecting a censorship-resilient BC as the BoB or the target BC. However, these protocols still have two disadvantages. First, when a CCDI process operates under these protocols, it is still vulnerable to withholding attacks. This is because these protocols can only roll back the whole operations of the CCDI process when withholding attacks occurs, rather than further improving the quality of the atomicity property to prevent from denying CCDI services. Second, they hardly preserve the independence property of involved BCs, as these BCs need to store information to validate other BCs' proofs or BoBs' statements.

Based on the above analysis, we ask the second research question: *whether there exists a protocol that can (i) progress CCDI services even in the presence of withholding attacks, (ii) preserve BCs' independence, and (iii) offer higher efficiency, regarding the metrics of communication, computation, memory and latency, than the state-of-the-art CCDI protocols?*

B. Contributions of This Paper

This study is devoted to providing solutions to the aforementioned two open research questions. The overarching contribution of this paper is to propose new protocols for enabling more secure and efficient data interoperability among multiple BCs. The contributions of this paper are 4-fold:

1. To propose a new CCTE protocol. Compared with the CCTE protocol in [3], the proposed protocol has 4 advantageous features: First, it does not need relay nodes. As a result, the protocol not only leads to a less memory cost compared with the protocol in [3] but also makes the CCTE process only involve two token transfer transactions, one less than the protocol in [3]. Second, the two token transfer transactions are executed in parallel, offering greater efficiency compared with the protocol in [3] that requires sequential execution of the three token transfer transactions. Third, unlike the protocol in [3] which assumes a 1:1 exchange rate for different token types, or the protocols [5-10] that assume pre-decided exchange rates, the proposed protocol is based on an Automatic Market Maker mechanism [32] and the cross-chain token claim scheme [11] with some modifications – with this design, the protocol can be used to provide practical exchange rates without compromising the BCs' independence. Last, it can complete all token transfer transactions even in the presence of censorship and withholding attacks; in contrast, the protocol in [3] needs to roll back all CCTE operations to preserve its atomicity property, leading to the denial of CCDI services.

2. To propose a new CCDI protocol. It only costs three transaction execution phases to complete a CCDI process, and our solution is optimal compared with the current protocols [12-30]. Specifically, after locking all SD items involved in a CCDI process (Phase 1), their metadata are submitted to a BoB, and the BoB then executes the CCDI operations to update these metadata (Phase 2). The updated metadata is finally sent back to their BCs (Phase 3). Since storing and updating metadata is

[2] The fungibility property ensures that token transfer transactions within a CCTE process are indistinguishable from standard token transfer transactions.

more computationally efficient than creating a SD item in a BC, this design is more lightweighted than the widely adopted "mint or claim"-based mechanisms [33, 34].

Furthermore, we propose a threshold-signature-based "transfer and in place data update" mechanism and integrate it into the CCDI protocol. The mechanism preserves each participating BC's independence and offers more efficient performance than the "lock and claim" mechanisms [33, 34]. Specifically, all BoB nodes collectively create an account represented by a threshold public key, and the account is registered on each participating BC. At the start of a CCDI process, the relevant SD items in a BC are transferred to the account for locking operation. Once the BoB yields updated metadata, the updated metadata plus a threshold signature [35] are submitted back to the BC. The BC verifies the signature and updates the locked SD items according to the updated metadata. Since the threshold public key of the account can verify the signature, no external data (e.g., BoB's or other BCs' root hashes) are required, thereby retaining each BC's independence. Moreover, a transaction containing a single threshold signature can simultaneously update multiple SD items in a BC, which further boosts the efficiency of the CCDI protocol.

As another critical part of the proposed protocol, a state-data migration scheme and a malicious user nodes elimination scheme are proposed and integrated into the CCDI protocol. The former scheme enables a user to migrate its SD item from a censorship-prone BC to a censorship-resilient BC without compromising either BC's independence. The latter scheme identifies the users that launch withholding attacks. By eliminating the identified users, SD items of honest users can be further updated, instead of rolling back all operations involved in a CCDI process to preserve atomicity property.

3. To establish an analysis framework to prove the proposed CCTE and CCDI protocols satisfy the 3 properties that are outlined in [31] as necessary properties of a cross-chain protocol, i.e., atomicity, isolation and durability[3]. Besides these three properties, we also prove the proposed CCTE protocol is with the fungibility and script-minimum properties[4]. The fungibility property is essential for mitigating censorship attacks, while the script-minimum property enhances the compatibility of a CCTE protocol by enabling BCs that do not support smart contracts to participate in the CCTE process.

4. To implement the proposed protocols on Ganache [36] and Remix-IDE Ethereum [37]. Comprehensive case studies are conducted to evaluate the protocols by comparing with the distinguished protocols [3, 10, 26, 28]. The evaluation results demonstrate that the proposed protocols are with lower communication, computation, memory and latency cost.

The rest of this paper is organized as follows. Section II introduces some basic concepts related to this research. Section III provides an overview of the proposed protocols. Technical details of the proposed CCTE and CCDI protocols are presented in Sections IV and V, respectively. Simulation is reported in Section VI, followed by a review of related work in Section VII. Section VIII concludes the paper and discusses possible future work.

II. Basic Concepts

This section introduces some basic concepts that are preliminaries of understanding the proposed protocols.

A. Blockchain

A BC [1] is a distributed ledger comprising a sequence of blocks, where each block cryptographically links to its predecessor. A BC could be associated with the independence property defined below.

Independence property: *For a BC with a set of verification nodes $\mathcal{V}$. The independence property of the BC holds if every node $v \in \mathcal{V}$ only stores the state data of the BC and does not store the state data of any other BCs.*

B. Consensus Protocols

A consensus protocol [38] refers to as a set of instructions executed by a BC's nodes to achieve agreement on a block of transactions. A secure consensus protocol is considered to have safeness and liveness properties at the same time [1]. The safeness property ensures all transactions are correctly executed and are executed in a globally consistent order. The liveness property ensures that all the valid transactions submitted to a BC will eventually be stored into the BC.

The consensus protocols can be generally divided into deterministic protocols and probabilistic protocols [39]. With deterministic consensus protocols, once a block is generated, the transactions within it become immutable. In contrast, according to the common prefix property [40], probabilistic protocols ensure that once a block is extended by a sufficient number of subsequent blocks, transactions included in this block becomes immutable with high probability.

C. Smart Contracts

A smart contract is a code container deployed in BCs, which contains deterministic algorithms [41]. The execution of a smart contract can be represented as:

$$S_{sc}^{new} \leftarrow SC(S_{sc}, Txs(SD, params)) \tag{1}$$

where $SC(\cdot)$ denotes the smart contract; S_{sc} is the current state information of $SC(\cdot)$; $Txs(SD, params)$ represents the transactions which invoke $SC(\cdot)$, and each transaction contains state data SD and necessary parameters (denoted as $params$), e.g., signatures; S_{sc}^{new} is the output generated by the smart contract, e.g., the updated state of $SC(\cdot)$. The equation of (1) represents a state transition of $SC(\cdot)$ from S_{sc} to S_{sc}^{new}.

D. Cross-Chain Data Interoperability Protocol

A CCDI protocol defines a set of executable instructions. Execution of a CCDI protocol will lead to a state transition of all the BCs involved in a CCDI process. A CCDI process defined by a CCDI protocol includes the following elements:

[3] The isolation property ensures that transactions that are executed concurrently do not affect each other. The durability property ensures that a transaction can only be executed if the last transaction has correctly completed and has produced permanent result.

[4] The script-minimum property ensures that only signature verification scripts are required to execute token transfer transactions.

(i) A set of BCs $\boldsymbol{\mathcal{B}} = \{BC_1, BC_2, \ldots\}$ and a group of users $\boldsymbol{\mathcal{U}} = \{U_1, U_2, \ldots\}$. Both are considered as participants of the CCDI process.

(ii) A list of transactions to be executed denoted as $\boldsymbol{\mathcal{T}} = \{Tx_1, Tx_2, \ldots\}$. Each transaction in $\boldsymbol{\mathcal{T}}$ is generated by a specific user from $\boldsymbol{\mathcal{U}}$ and it will be executed in a specific BC from $\boldsymbol{\mathcal{B}}$.

(iii) Execution phases of the CCDI process, denoted as $\boldsymbol{\mathcal{P}} = \{P_1, P_2, \ldots\}$. The phases are executed in a sequential manner, and each phase involves the execution of one or more transactions. To execute a phase, all the transactions involved can be concurrently executed.

E. Merkle Proofs

A Merkle proof [16] certifies the existence of a transaction in a block. Key operations for Merkle proofs include:

(i) $Gen_{proof}(Tx, BH)$. It means the creation of a Merkle proof for a transaction Tx in a block with its block header BH.

(ii) $Val_{proof}(P, Tx, R_{BH})$. It represents the verification of a Merkle proof P. R_{BH} is a root hash of a block header BH. The verification operation returns true if Tx is in the block with a header of BH; it returns false otherwise.

F. Threshold Signature

A threshold signature (TS) [35] is used to certify that a message m is agreed by a group of nodes. Consider a blockchain comprising $M = 3f + 1$ nodes, where f is the number of Byzantine consensus nodes. If m is valid, each node independently signs m to generate a TS share. A TS is then constructed by combining at least $2f + 1$ distinct TS shares generated by different nodes. Finally, the validity of the resulting TS is verified using a threshold public key.

III. System and Threat Models, Assumptions and Goals

This section provides an overview of the system to which the proposed protocols apply, together with the threat models, assumptions and design goals of the protocols.

A. System Model

Our system model is based on a blockchain and blockchains-based multi-blockchains system (BoBMBS). A BoBMBS consists of a BC acting as a BoB and multiple BCs (called the external BCs). We denote the number of external BCs in a BoBMBS as N and denote the external BCs as BC_i, $i \in [N]$, where $[N]$ is short for the set $\{1,2, \ldots, N\}$. For BC_i or the BoB, it progresses in consecutive epochs, with the e_i-th or e_B-th epoch indexed by e_i or e_B ($e_i \in \mathbb{N}, e_B \in \mathbb{N}$). At the end of each epoch, a block is generated in a BC. All BCs in the system operate under the semi-synchronous network model [38] and the adaptive corruption model [39]. In semi-synchronous networks, there exists an unknown time duration Δ on message delivery delays, so a block generation time is not fixed. In the adaptive corruption model, an adversary may, at any time point during the execution of a consensus protocol, corrupt up to f consensus nodes in a BC.

B. Threat Model

Two attack scenarios (i.e., censorship attacks and withholding attacks) are considered in the BoBMBS system, aiming to deny CCDI services or make a CCDI protocol violate its atomicity property.

Censorship attacks. In a normal case, a transaction Tx sent to a blockchain BC_i ($i \in [N]$) in epoch e_i is immutably stored in BC_i in epoch $e_i + \kappa_i + 1$, where $\kappa_i \geq 0$ is a parameter of the common prefix property [40] of BC_i. A censorship attack delays the immutable inclusion of Tx in BC_i to epoch $e_i + \kappa_i + 1 + D_i$, where D_i is a finite positive integer.

Withholding attacks. In a CCDI protocol, a normal case is that the user node participating in a CCDI process should send its transaction Tx to a specified BC (see Section II-D) before a deadline defined by a CCDI protocol. A withhold attack occurs when the user node either submits Tx after this deadline or fails to submit it throughout the CCDI process.

C. Protocols Assumption and Design Goals

Assumptions. The proposed CCDI protocols work in the BoBMBS. The following assumptions are applied to ensure the security of the protocols:

(i) A deterministic consensus protocol is used in the BoB.

(ii) The consensus nodes in the BoB store both the BoB's state data and the latest κ_i block headers of BC_i ($i \in [N]$).

(iii) Every BC and the BoB in the BoBMBS hold the safeness and liveness properties.

(iv) The BoB and some BCs in the BoBMBS are censorship-resilient due to their protocols' design [41-43].

Goals. It is expected that the proposed protocols can achieve the following the following design goals:

The independence property of each external BC in the BoBMBS is preserved when these BCs involve in the CCDI processes defined by the proposed protocols.

The protocols hold the atomicity, isolation, and durability properties. These properties are defined as follows:

Atomicity: *In a CCDI process, the states of all the participants either successfully transit to new states or remain unchanged.* Note that the atomicity property does not apply to the changes of a participant's state that is caused by the operations unrelated to the CCDI process.

Isolation: *In a phase of the CCDI process when multiple transactions need to be concurrently executed, each transaction must be executed independently.*

Durability: *When a transaction Tx_1 must be executed in prior to another transaction Tx_2, the change of the BoBMBS's state caused by Tx_1 must be permanently and immutably stored in the corresponding BC before executing Tx_2.*

Besides the above properties, the proposed CCTE protocol (see Section IV) holds two additional properties:

Fungibility: *For observers except for transaction formation agents, token receivers and the BoB, they cannot distinguish a token transfer transaction in a CCTE and a standard* *tokentransfer transaction*[5].

Script-minimum: *An external BC only needs signature verification scripts to participate in a CCTE process.*

IV. Cross-Chain Token Exchange Protocol

The proposed CCTE protocol (denoted as Π_{CCTE}) is designed for CCTE (atomic swaps) scenarios where a sender intends to use its tokens of type $Token_i$ to exchange for a receiver's x_j tokens of type $Token_j$, and these two types of tokens circulate on BC_i and BC_j, respectively. The CCTE process defined in Π_{CCTE} involves 5 basic transactions and 5 participants. The transactions include a cross-chain transaction (CCT), two native transactions (denoted as NT_s and NT_r), and two proof transactions (denoted as PT_s and PT_r). The participants include a sender, a receiver, the two blockchains BC_i and BC_j, and the BoB. In some situations, an additional cross-chain transaction CCT_V and at most two fraud-proof transactions (each denoted as T_{FP}) (see Phase 3 of this section) will be needed. The type of the token circulating in the BoB is denoted as $Token_B$.

As illustrated in Fig. 1, the CCTE process comprises of 3 phases: firstly, the sender sends the CCT to the BoB for activating the CCTE process. Secondly, the sender sends the native transaction NT_s to BC_i and the BoB for transferring its tokens with type $Token_i$ to the receiver's address in BC_i; in the same time, the receiver sends the native transaction NT_r to BC_j and the BoB for transferring its tokens with type $Token_j$ to the sender's address in BC_j. Phase 3 will be executed for security purposes, in which the proof transactions PT_s and PT_r will be sent to the BoB. The transaction execution logics are supported by four smart contracts deployed in the BoB. Table I provides a brief description on their functionalities, while their roles will be explained later in this section.

A. Workflow of Π_{CCTE}

The overall procedure of CCTE is shown in Algorithm 1. To defend against censorship and withholding attacks, before the CCTE process, the sender and the receiver need to deposit a certain amount of $Token_B$ (denoted the amounts as $x_{s,B}$ and $x_{r,B}$, respectively) in the BoB through the smart contract SC_D (ln 01). The CCTE process is performed with the three phases:

Phase 1: The sender generates a cross-chain transaction CCT and sends it to the BoB to invoke the smart contract SC_T (ln 02). SC_T requests two exchange rates $E_{i,j}$ and $E_{B,j}$ from the smart contract SC_{AMM} (ln 03), where $E_{i,j}$ is the exchange rate between $Token_i$ and $Token_j$ and $E_{B,j}$ is the exchange rate between $Token_B$ and $Token_j$. These two rates are determined following an exchange rate determination scheme, which will be elaborated in Section IV-B. Based on the exchange rates, SC_T calculates the amount of $Token_i$ or $Token_B$ needed for exchanging x_j $Token_j$ (ln 04).

To validate the cross-chain transaction CCT, the smart contract SC_T checks whether the sender and the receiver have enough deposits of $Token_B$ by invoking the smart contract SC_D (ln 05). If the sender or the receiver does not have enough deposits, the CCT is discarded, which can prevent participators from launching withholding attacks afterwards (ln 09). If they have, SC_T generates two instructions IN_s and IN_r (lns 06 and 07): IN_s is for the sender to transfer x_i $Token_i$ to the receiver in BC_i; IN_r is for the receiver to transfer x_j $Token_j$ to the sender in BC_j. Aiming at quickly terminating the CCTE process, SC_T sets a deadline before which proof transactions on IN_s and IN_r must be included to the BoB (ln 08).

Phase 2. The sender and the receiver generate native transactions $NT_s(IN_s)$ and $NT_r(IN_r)$ for the two instructions IN_s and IN_r, respectively. They send $NT_s(IN_s)$ and $NT_r(IN_r)$ to BC_i and BC_j, respectively. In the meantime, they send $NT_s(IN_s)$ and $NT_r(IN_r)$ to the BoB for censorship and withholding resistance purposes (lns 10 and 11), as the two sending operations enable the consensus nodes in the BoB to submit $NT_s(IN_s)$ and $NT_r(IN_r)$ to corresponding BCs. If the sender and the receiver are honest and no attack exists, the CCTE process terminates at the end of Phase 2; otherwise, Phase 3 is needed:

Phase 3. If the native transactions $NT_s(IN_s)$ and $NT_r(IN_r)$ are stored in the corresponding BCs, the sender and the receiver generate Merkle proofs P_s and P_r for $NT_s(IN_s)$ and $NT_r(IN_r)$, respectively (lns 12 and 13). Each Merkle proof and the corresponding native transaction are packaged into a proof transaction. The proof transactions are sent to the BoB to invoke the smart contract SC_{MPV} (lns 14 and 15). SC_{MPV} validates the received proofs and records the valid Merkle proofs (lns 16-18). Then, there are two situations to confirm the CCT:

Situation 1: If SC_{MPV} receives the two valid proofs, the CCTE process completes (lns 19 and 20).

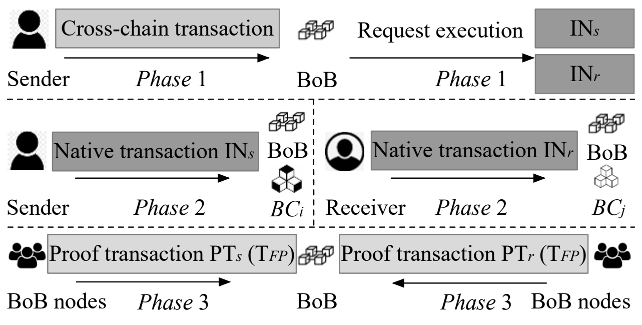

Fig. 1 The transactions of Π_{CCTE} in each phase.

TABLE I
Smart Contracts Used in Π_{CCTE}

Notation	Functionality
SC_D	Lock $Token_B$
SC_T	Execute cross-chain transactions
SC_{AMM}	Calculate exchange rates
SC_{MPV}	Validate received Merkle proofs

[5] The standard token transfer transaction contains the sender's and the receiver's public keys, the sender's signature, the token's type and the amount to be transferred.

Algorithm 1 CCTE Procedures

Notations

$*= i$ or j; e_*: BC_*'s epoch index in which CCT is sent to the BoB; $PK_{s,i}$, $PK_{s,j}$ and $PK_{s,B}$: IDs of the senders in BC_i, BC_j and the BoB, respectively; $PK_{r,i}$, $PK_{r,j}$ and $PK_{r,B}$: IDs of the receiver in BC_i, BC_j and the BoB, respectively; $\bar{\kappa}$: the secure parameter ($\bar{\kappa} \geq 2$); $Max()$: a function returning the largest value among the inputted values; $Min()$: a function returning the smallest value among the inputted values; $BH_*^{e_*'}$: the header of BC_*'s block generated in epoch e_*'; $R(BH_*^{e_*'})$: the root hash of $BH_*^{e_*'}$. FP: the fraud-proof generated by consensus nodes of the BoB. e_*^c: the current epoch index of BC_*. $x_{s,i}$: the number of $Token_i$ in the sender's account in BC_i; $x_{s,j}$: the number of $Token_j$ in the receiver's account in BC_j.

CCTE instance: exchange for x_j $Token_j$ using tokens with type $Token_i$.

```
01  $SC_D$: $PK_{s,B}$ =[$Token_B$: $x_{s,B}$]; $PK_{r,B}$ =[$Token_B$: $x_{r,B}$]
Phase 1 – One cross-chain transaction execution
02  $PK_{s,B} \rightarrow CCT(BC_i, BC_j, PK_{s,i}, PK_{s,j}, PK_{r,i}, PK_{r,j}, PK_{r,B}, x_j,$
    $Token_i, Token_j, SC_T) \rightarrow$ BoB
03  $E_{i,j} = C_{AMM}(Token_i, Token_j)$; $E_{B,j} = SC_{AMM}(Token_B, Token_j)$
04  $x_i = x_j/E_{i,j}$; $x_B = x_j/E_{B,j}$
05    $SC_T \rightarrow SC_D$: if $x_{s,B} > \bar{\kappa} \times x_B$ and $x_{r,B} > \bar{\kappa} \times x_B$:
06      $SC_T$: $IN_s$ =(From: $PK_{s,i}$, To: $PK_{r,i}$, Transfer: $x_i$ $Token_i$)
07      $IN_r$ =(From: $PK_{r,j}$, To: $PK_{s,j}$, Transfer: $x_j$ $Token_j$)
08      $Max(e_i + \bar{\kappa} + \kappa_i, e_j + \bar{\kappa} + \kappa_j)$; $e_* + \bar{\kappa}$: the epoch index of $BC_*$
        when $CCT$ is stored
09    Else: discard $CCT$ of ln 02 and terminate
Phase 2 – Two native transactions execution concurrently
10  Sender: $PK_{s,i} \rightarrow NT_s(IN_s) \rightarrow BC_i$ and BoB
11  Receiver: $PK_{r,j} \rightarrow NT_r(IN_r) \rightarrow BC_j$ and BoB
Phase 3 – Two proof transactions execution concurrently
12  $P_s = Gen_{proof}(NT_s(IN_s), BH_i^{e_i'})$
13  $P_r = Gen_{proof}(NT_r(IN_r), BH_j^{e_j'})$
14  $PT_s(P_s, NT_s(IN_s), R(BH_i^{e_i'}), SC_{MPV}) \rightarrow$ BoB
15  $PT_r(P_r, NT_r(IN_r), R(BH_j^{e_j'}), SC_{MPV}) \rightarrow$ BoB
16  $SC_{MPV}$: (i) $CCT \in SC_T$;(ii) $e_*' > e_* + \bar{\kappa}$; (iii) $Val_{proof}(P, NT, R(BH_*^{e_*'}))$
    If true is returned for all the three checking operations:
17    $P \in SC_{MPV}$
18  Else: discard $P$
19  Situation 1: If $(P_s, P_r) \in SC_{MPV}$
20    $SC_{MPV}$: Confirm $CCT$ of ln 02
21  Situation 2: If $P_s$ and/or $P_r \notin SC_{MPV}$ and
    $Min(e_i^c, e_j^c) \geq Max(e_i + \bar{\kappa} + \kappa_i, e_j + \bar{\kappa} + \kappa_j)$
22    The nodes of the BoB: $CCT_V(SC_{MPV}) \rightarrow$ BoB
23    $SC_{MPV}$: execute ln 21; if true is returned, lns 24-29; else: ln 30
24      Sub-situation 1: If $NT_s(IN_s) \notin BoB$: $SC_{MPV}$: (From: $PK_{s,B}$, To:
        $PK_{r,B}$, Transfer: $x_B$ $Token_B$), confirm $CCT$ of ln 02
25      Sub-situation 1: If $NT_r(IN_r) \notin BoB$: $SC_{MPV}$: (From: $PK_{r,B}$, To:
        $PK_{s,B}$, Transfer: $x_B$ $Token_B$), confirm $CCT$ of ln 02
26      Sub-situation 2: If $NT_s(IN_s) \in BoB$: execute ln 14 and invoke ln 19
27      Sub-situation 2: If $NT_r(IN_r) \in BoB$: execute ln 15 and invoke ln 19
28      Sub-situation 3: If $x_{s,i} < x_i$: generate $T_{FP}$ to invoke lns 16 and 24
29      Sub-situation 3: If $x_{r,i} < x_j$: generate $T_{FP}$ to invoke lns 16 and 25
30    $SC_{MPV}$: Discard $CCT_V(SC_{MPV})$
```

*Note: For simplicity, the signatures contained in the transactions are omitted. This applies to the rest of this paper.

Situation 2: If one or both valid proofs fail to be recorded in SC_{MPV} by the deadline defined in ln 08, a transaction CCT_V is then generated by a node of the BoB to invoke SC_{MPV} (lns 21 and 22). SC_{MPV} then checks whether the situation defined in ln 21 is true (ln 24). If it is false, CCT_V is discarded (ln 30). Otherwise, the following 3 sub-situations may happen:

Sub-situation 1: If either $NT_s(IN_s)$ or $NT_r(IN_r)$ fails to appear in the BoB by its deadline (ln 08) due to withholding attacks, SC_{MPV} transfers the sender's (or the receiver's) deposited tokens with type of $Token_B$ to the counterparty's account in the BoB and the CCTE process completes (ln 24 or ln 25).

Sub-situation 2: If $NT_s(IN_s)$ (or $NT_r(IN_r)$) is recorded in the BoB, $f+1$ pre-defined consensus nodes of the BoB obtain $NT_s(IN_s)$ (or $NT_r(IN_r)$) and sends it to BC_i (or BC_j). If $NT_s(IN_s)$ (or $NT_r(IN_r)$) is stored in BC_i (or BC_j), the $f+1$ consensus nodes generate its proof transaction (ln 14 or ln 15) and send it to the BoB for further execution (ln 26 or ln 27).

Sub-situation 3: Before $NT_s(IN_s)$ (or $NT_r(IN_r)$) is stored in BC_i (or BC_j), if any of the $f+1$ nodes finds that the balance of the sender's $Token_i$ (or the receiver's $Token_j$) on BC_i (or BC_j) falls below x_i (or x_j), it generates a fraud-proof transaction T_{FP} with a Merkle proof and send T_{FP} to the BoB (ln 28 or ln 29).

B. Token Exchange Rate Determination Scheme

We design a Token exchange rate determination scheme to enable the smart contract SC_{AMM} to promptly determine the exchange rates between two types of tokens circulating in the BoBMBS. The scheme comprises two components: an Automated Market Maker (AMM) and a Cross-chain Token Claim Protocol (Π_{CCTC}). We will present the two components first, followed by the procedure of the token exchange rate determination scheme.

Automated Market Maker. AMM uses a Liquidity Pool (LP) and mathematical algorithms to calculate exchange rate(s) between two types of tokens in a BC. The LP is deployed in the BoB, and it can be considered as a marketplace for token providers to save different types of tokens. For illustration, AMM considers the Balancer protocol [32] to calculate the exchange rate. For simplicity, we assume that there are N types of tokens in the system, where each BC has one type of token. During the initiation of the LP, the AMM assigns a pre-defined weight to each token type (denoted as w_i, $i \in [N]$) with a condition of $\sum_{i=1}^{N} w_i = 1$. It then determines the exchange rate between two different types of tokens as:

$$E_{i,j} = \frac{B_j}{B_i} \times \frac{w_i}{w_j} \tag{2}$$

where $E_{i,j}$ is the exchange rate between $Token_i$ and $Token_j$; B_i and B_j are the quantity of the two types of tokens in the LP. B_i and B_j will be quantified in the LP based on the cross-chain token claim protocol (denoted as Π_{CCTC}):

Cross-chain token claim protocol. Based on the token exchange rates generated from the AMM, Π_{CCTC} is designed to facilitate a token provider to save a specific type of token from the corresponding external BC to the LP. Π_{CCTC} is based on the cross-chain token claim scheme in [11] with modifications. Considering a token provider intends to save x tokens with the type of $Token_*$ to the LP, the protocol works as follows:

(i) The BoB validating nodes jointly create a threshold public key-based address [35] in each external BC BC_i. Denote the address as TPK_Addr_i, $i \in [N]$.

(ii) The token provider sends the x tokens with the type of $Token_*$ to an address TPK_Addr_* for locking operation.

(iii) The token provider generates a Merkle proof for the token transfer operation in (ii). Based on the proof, SC_M mints x $token_*$ in the BoB and transfers the minted tokens to the token provider's address in the BoB.

(iv) The token provider transfers the x $Token_*$ from its address in the BoB to the AMM smart contract SC_{AMM}.

Procedure of token exchange rate determination. With Π_{CCTC}, any types of tokens from an external BC can circulate in the BoB. The token exchange rate is determined via 3 steps:

(i) Smart contract deployment. The AMM is implemented in SC_{AMM} that is deployed in the BoB. There are two data structures in the LP of SC_{AMM}: $\mathbf{LP}_V$ and $\mathbf{LP}_{KV}$. $\mathbf{LP}_V$ stores N pre-defined weighting values, one for a type of token. $\mathbf{LP}_{KV}$ stores N key-value pairs. For each pair, the key is the ID of a specific token type, and the value is the number of tokens with this type.

(ii) Initialize the LP. SC_{AMM} invokes a function (denoted as $\mathcal{F}_{VI}$) to initialize the N weights in $\mathbf{LP}_V$:

$$(w_1, \dots, w_N) = \mathcal{F}_{VI}(Token_1, \dots, Token_N) \qquad (3)$$

(iii) Determination of the token exchange rate. Token providers contribute their claimed tokens to SC_{AMM}. SC_{AMM} stores the key-value pairs of the tokens into $\mathbf{LP}_{KV}$. Once $\mathbf{LP}_V$ and $\mathbf{LP}_{KV}$ are constructed, (2) can be applied to calculate the exchange rate between any two types of tokens.

C. Security Analysis

This section is devoted to analyzing the security of the proposed CCTE protocol. The analysis is represented by the following theorems:

Theorem 1. *For the CCTE process defined in Π_{CCTE}, the independence and the script-minimum properties of each participating BC_i ($i \in [N]$) in the BoBMBS can be preserved.*

Proof. Consider an arbitrary participating BC_i ($i \in [N]$). In the second phase of the CCTE process, a token transfer transaction NT_s will be executed in BC_i. To execute NT_s, BC_i only need to check whether the sender of NT_s has sufficient tokens, which does not need the external state data. Thus, the independence property is held by BC_i. Since the execution of the transaction NT_s only requires BC_i to have the scripts for validating the signature in NT_s, the script-minimum property is also preserved. □

Theorem 2. *Under the corruption and network models defined in Section III-A, if every participating BC_i ($i \in [N]$) and the BoB holds the safeness and liveness properties, Π_{CCTE} will preserve the atomicity, isolation, durability properties.*

Proof. For the atomicity property: the CCTE process has two state transitions: (a) the sender consumes x_i $Token_i$ (or x_B $Token_B$) and obtains x_j $Token_j$ (or x_B $Token_B$). And (b) the receiver obtains x_i $Token_i$ (or x_B $Token_B$) and consumes x_j $Token_j$ (or x_B $Token_B$). There are three situations:

(i) Situation 1 – the CCT is invalid. In this situation, the CCT will be discarded and the CCTE process will terminate without any state transition (see lns 05 and 09 of Algorithm 1).

(ii) Situation 2 – the CCT is valid, and both the sender and the receiver are honest. Once the CCT is validated to be valid, two instructions (i.e., IN_s and IN_r in Algorithm 1) will be generated. Also, the honest sender and the honest receiver will send NT_s and NT_r to BC_i and BC_j, respectively. In the meantime, since NT_s and NT_r are also sent to the BoB before the deadline defined in Algorithm 1 (see ln 08), at least one of the $f+1$ nodes in the BoB will submit NT_s and NT_r to corresponding BCs. According to the liveness property, even if censorship attack is launched, NT_s and NT_r are guaranteed to be stored in corresponding BCs. As a result, successful state transitions will be achieved. However, if at least one participator (the sender and/or receiver) is malicious, it may launch a withholding attack (see lns 24 and/or 25 of Algorithm 1), or the balance in its account may not enough (see lns 28 and/or 29 of Algorithm 1). Then, Situation 3 will be triggered:

(iii) Situation 3 – the CCT is valid, and the sender and/or the receiver are malicious. According to lns 21-29 of Algorithm 1, each missed native transaction or proof transaction (due to withholding attacks), or the presence of valid fraud-proof transactions (due to the insufficient balance) will lead to a transfer of $Token_B$ for completing the unfinished token transfer operations. Thus, the state transition will be successful. As a result, the atomicity property is preserved.

Regarding the isolation property: the two native transactions will be executed concurrently in BC_i and BC_j, while the two proof transactions PT_s and PT_r will be executed concurrently in the BoB. Due to the independence of BC_i and BC_j, each is unaware of the other's native transaction execution. Similarly, the execution of PT_s and PT_r in the BoB is independent – that is, the verification and execution of PT_s do not require any information from PT_r, and vice versa. Therefore, the isolation property is preserved.

To ensure the durability property, (i) the native transaction (such as NT_s) can only be executed after the CCT has been stored in the BoB and (ii) the proof transaction (such as PT_s) can only be executed after the native transaction NT_s are stored in BC_i.

To ensure (i), if NT_s is stored in BC_i before the CCT has been stored in the BoB, i.e., $e_i^{'} \leq e_i + \bar{\kappa}$ (see ln 16 of Algorithm 1), PT_s will be discarded according to ln 18 of Algorithm 1. To ensure (ii), lns 06 and 18 also guarantee that if NT_s is not stored in BC_i (i.e., $Val_{proof}(P_s, NT_s, R(BH_i^{e_i^{'}}) \neq 1)$, PT_s will be discarded according to ln 18 of Algorithm 1.

When CCT_V is required to be sent to the BoB (see ln 22 of Algorithm 1), to ensure the durability property, CCT_V can only be executed after the deadline defined in ln 08 of Algorithm 1. The checking operation in ln 21 guarantees that if CCT_V is sent to the BoB before the deadline, it will be discarded.

When a fraud transaction containing a Merkle proof is sent to the BoB (see lns 28 and 29 of Algorithm 1), the BoB will verify that whether the Merkle proof can attest that the corresponding native transaction (NT_s or NT_r) failed to be stored on its target BC because the on-chain balance of its sender's account fell below the required threshold. As a result, the fraud-proof transaction can only be executed when the Merkle proof is valid. Therefore, every transaction in Π_{CCTE} is executed following the criteria defined in the durability property. □

Theorem 3. *every native transaction in the CCTE process defined by Π_{CCTE} preserves the fungibility property.*

TABLE II
Smart Contracts Used in Π_{CCDI}

Notation	Functionality
SC_{BG}	Deploy battle games, create and update state data items
SC_{C}	Validate cross-chain transactions
$SC_{\mathcal{F}}$	Implement CCDI applications

Proof. In each external BC, only native transactions that carry token transfer instructions will be executed. Since the instructions include only the sender' and the receiver's IDs, the token type and the token transfer amount, they cannot be distinguished from standard token transfer transactions. Therefore, the fungibility property is preserved. □

V. Cross-Chain Data Interoperability Protocol

The proposed cross-chain data interoperability CCDI protocol Π_{CCDI} operates under the BoBMBS including the BoB and N external blockchains BCs and defines a more generic CCDI process $\mathcal{P}_{CCDI}$. Without loss of generality, we consider that X user nodes register to participate in $\mathcal{P}_{CCDI}$. Each user node has a unique state data item, and the x-th ($x \in [X]$) user node (denoted as U_x) is from BC_i, where $i \equiv x \bmod N$. Additionally, Π_{CCDI} defines that each BC configures a relay group that comprises of $(f+1)$ relay nodes randomly selected from the M consensus nodes in the BoB. Π_{CCDI} further defines 3 smart contracts for $\mathcal{P}_{CCDI}$. SC_C and $SC_{\mathcal{F}}$ are deployed in the BoB, and SC_{BG} is deployed in every external blockchain BC_i ($i \in [N]$). Their functionalities are shown in Table II.

To illustrate a concrete CCDI application, we take battle games as an example. A battle game includes a Player versus Player (PvP) instance, which represents the scenario that two players battle with each other. A player uses a unique state data item representing its game character, which includes a role name, an attack value, and an experience value. In a PvP, a player's score is the sum of its character's attack and experience values. The player with a higher score will win the battle. These logics of battle games will be coded in the smart contracts SC_{BG} and $SC_{\mathcal{F}}$. In the following, we further present two building blocks of the CCDI protocol Π_{CCDI}.

A. Two Building Blocks of Π_{CCDI}

The first building block is an abstract CCDI application smart contract $SC_{\mathcal{F}}$ that can host any types of CCDI applications. In this paper, $SC_{\mathcal{F}}$ deploys the logic of battle games. To invoke $SC_{\mathcal{F}}$, a set of state data items' metadata (denoted as $\{SD_1, \dots, SD_x, \dots\}$) inputs into $SC_{\mathcal{F}}$ which outputs a new set $\{SD_1^{new}, \dots, SD_x^{new}, \dots\}$ according to the logic of battle games:

$$\{SD_1^{new}, \dots, SD_x^{new}, \dots\} = SC_{\mathcal{F}}(SD_1, \dots, SD_x, \dots) \quad (4)$$

where $\{SD_1^{new}, \dots, SD_x^{new}, \dots\}$ is the updated versions of $\{SD_1, \dots, SD_x, \dots\}$. Since $SC_{\mathcal{F}}$ only consumes metadata, the data format unification mechanisms [22, 28] can be applied to the received metadata, harmonizing their data formats before submitting them to $SC_{\mathcal{F}}$. Upon outputting the updated metadata, they can be converted back to their original data formats.

The second building block is a threshold-signature-based

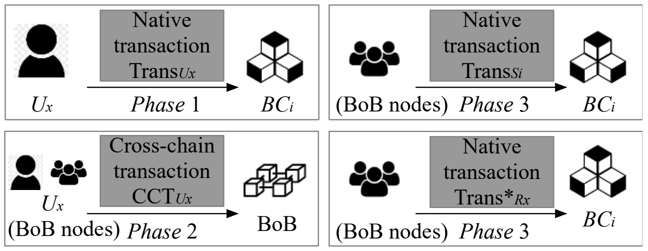
Native transaction Trans$_{Ux}$
U_x
Phase 1
BC_i
Native transaction Trans$_{Si}$
(BoB nodes)
Phase 3
BC_i
Cross-chain transaction CCT$_{Ux}$
U_x
(BoB nodes)
Phase 2
BoB
Native transaction Trans*$_{Rx}$
(BoB nodes)
Phase 3
BC_i

Fig. 2 The transactions of Π_{CCDI} in each phase.

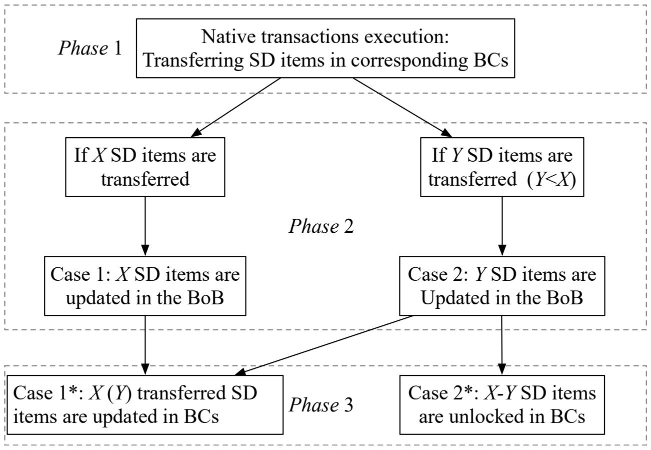
Phase 1
Native transactions execution: Transferring SD items in corresponding BCs
If X SD items are transferred
If Y SD items are transferred (Y<X)
Phase 2
Case 1: X SD items are updated in the BoB
Case 2: Y SD items are Updated in the BoB
Case 1*: X (Y) transferred SD items are updated in BCs
Phase 3
Case 2*: X-Y SD items are unlocked in BCs

Fig. 3 Flowchart of $\mathcal{P}_{CCDI}$.

"transfer and in place data update" mechanism. Specifically, the nodes of the BoB collectively generate a threshold public key PK, and the i-th group generates an account using PK in the blockchain BC_i ($i \in [N]$). The smart contract SC_{BG} then grants PK the ability to update the state data items that PK possesses. Thus, PK now can invoke SC_{BG} to update its state data items. The details of the "transfer and in place data update" mechanism are as below.

When a user node U_x in the blockchain BC_i registers to participate in a CCDI process $\mathcal{P}_{CCDI}$, it first sends its state data item (denoted as SD_x) to the threshold public key PK, which acts as the locking operation. When SD_x is updated in the BoB through the smart contract $SC_{\mathcal{F}}$, the nodes in the BoB generate a transaction containing a threshold signature TS and the updated SD_x (denoted as SD_x^{new}). The transaction is then sent to BC_i to invoke smart contract SC_{BG}. If PK in SC_{BG} validates TS, SD_x is updated to SD_x^{new} in BC_i and sent back to U_x's account.

Next, we present the details of the protocol Π_{CCDI} (as shown in Fig. 2) as below.

B. Phase 1 – Preparation

Let the CCDI process be initiated at the beginning of the Bo B's epoch indexed by e_B and a deadline is set to be the end of the epoch $e_B + \kappa \times Max(\kappa_i)$, ($i \in [N]$), where κ is a secure parameter to ensure that every external blockchain BC_i has passed κ_i epochs since the beginning of epoch e_B. In epoch e_B, every user node U_x ($x \in [X]$) then sends a native transaction $Trans_{Ux}$ to the threshold public key PK in BC_i:

$$Trans_{Ux} = < SD_x, PK_x^i, PK, SC_{BG} > \quad (5)$$

where PK_x^i represents the account address of U_x in BC_i. BC_i

checks whether PK_x^i owns the state data item SD_x. If not, $Trans_{Ux}$ is discarded; otherwise, BC_i transfers SD_x to PK and creates a key-value pair $< H(Trans_{Ux}), (SD_x, PK_x^i) >$ in the smart contract SC_{BG}, where $H(\cdot)$ is a hash function.

C. Phase 2 – Execution

If $Trans_{Ux}$ is stored in BC_i before epoch $e_B + \kappa \times Max(\kappa_i)$, a Merkle proof (denoted by P_x) and a cross-chain transaction (denoted as CCT_{Ux}) are generated:

$$CCT_{Ux} =< Trans_{Ux}, BC_i, P_x, SC_C > \qquad (6)$$

The user node U_x and the $f+1$ replay nodes in the i-th relay group ($i \in [N]$) send CCT_{Ux} to the BoB to invoke the smart contract SC_C. After receiving CCT_{Ux}, SC_C checks whether it has stored the state data item SD_x that is contained in the $Trans_{Ux}$'s field of CCT_{Ux}. If yes, CCT_{Ux} is discarded; otherwise, SC_C checks:

❿ Whether the index of the BoB's current epoch (denoted as e_B^c) satisfies: $e_B^c < e_B + \kappa \times Max(\kappa_i)$.

❿ Whether P_x can prove that $Trans_{Ux}$ has been immutably stored in BC_i.

If CCT_{Ux} passes all the 2 checking operations, the state data item SD_x contained in the cross-chain transaction CCT_{Ux} is stored in the smart contract $SC_{\mathcal{F}}$; otherwise, CCT_{Ux} is discarded. As shown in Fig.3, $SC_{\mathcal{F}}$ performs one of the two operations:

Case 1: If all the X cross-chain transactions are stored in the smart contract $SC_{\mathcal{F}}$ by epoch $e_B + \kappa \times Max(\kappa_i)$, when $SC_{\mathcal{F}}$ stores the X-th state data item, it updates the X state data items according to (4) and the updated SD_x is denoted as SD_x^{new}. For the set of C updated state data items $\boldsymbol{S}_i = \{SD_{x_1}^{new}, \dots, SD_{x_c}^{new}, \dots, SD_{x_C}^{new}\}$ from the blockchain BC_i ($c \in [C]$ and $\forall c, i \equiv x_c \bmod N$), the M nodes of the BoB produce the threshold signature TS for the set $\boldsymbol{S}_i$.

Case 2: If there are some state data items among the X state data items not stored in the smart contract $SC_{\mathcal{F}}$ by epoch $e_B + \kappa \times Max(\kappa_i)$, $SC_{\mathcal{F}}$ identifies the number of the user nodes' state data items that have been received before epoch $e_B + \kappa \times Max(\kappa_i)$ (denoted as Y). If $Y > 1$, $SC_{\mathcal{F}}$ updates the Y state data items through (4). For the set of D updated state data items $\overline{\boldsymbol{S}}_i = \{SD_{x_1}^{new}, \dots, SD_{x_d}^{new}, \dots, SD_{x_D}^{new}\}$ from the blockchain BC_i ($d \in [D]$ and $\forall d, i \equiv x_d \bmod N$), the M nodes of the BoB produce the threshold signature TS for the set $\overline{\boldsymbol{S}}_i$. If $Y \le 1$, the M nodes of the BoB produce a threshold signature for the only received state data item.

If a valid cross-chain transaction CCT_{Ux} ($x \in [X]$) is sent to the BoB after epoch $e_B + \kappa \times Max(\kappa_i)$, the M nodes of the BoB also produce TS for the state data item SD_x contained in CCT_{Ux}, ensuring its unlocking during Phase 3 (see Case 2*).

D. Phase 3 – Settlement

The transferred state data items in Phase 1 are updated in their blockchains in this phase:

Case 1*: If the set $\boldsymbol{S}_i$ is generated in the BoB, a native transaction $Trans_{\boldsymbol{S}_i}$ is generated:

$$Trans_{\boldsymbol{S}_i} =< \boldsymbol{S}_i, TS, \boldsymbol{S}_{x_c}, SC_{BG} > \qquad (7)$$

where $\boldsymbol{S}_{x_c}$ is a set containing the C hash values $\{H(Trans_{Ux_1}), \dots, H(Trans_{Ux_C})\}$. After that, the $f+1$ replay nodes in the i-th group send $Trans_{\boldsymbol{S}_i}$ to the blockchain BC_i to invoke the smart contract SC_{BG}. SC_{BG} checks:

- Whether it has the key value pairs $< H(Trans_{Ux_1}), (SD_{x_1}, PK_{x_1}^i) >, \dots, < H(Trans_{Ux_C}), (SD_{x_C}, PK_{x_C}^i) >$.
- Whether the threshold public key PK validates TS.

If $Trans_{\boldsymbol{S}_i}$ passes all the two checking operations, $Trans_{\boldsymbol{S}_i}$ is regarded as valid; otherwise, $Trans_{\boldsymbol{S}_i}$ is discarded.

If $Trans_{\boldsymbol{S}_i}$ is valid, PK invokes SC_{BG} to update SD_{x_c} to $SD_{x_c}^{new}$ and sends $SD_{x_c}^{new}$ to the account address $PK_{x_c}^i$ ($c \in [C]$). After that, SC_{BG} deletes $< H(Trans_{Ux_c}), (SD_{x_c}, PK_{x_c}^i) >$.

The above procedures are also applied to $\overline{\boldsymbol{S}}_i$ in Case 2.

Case 2*: To respond to a valid cross-chain transaction CCT_{Ux} in Case 2, a native transaction $Trans_{Rx}^*$ is generated:

$$Trans_{Rx}^* =< SD_x, TS, H(Trans_{Ux}), SC_{BG} > \qquad (8)$$

The procedures for $Trans_{\boldsymbol{S}_i}$ is then applied to $Trans_{Rx}^*$ and SD_x will be directly sent back to U_x's account PK_x^i.

E. Attacks Resolution Mechanism

According to Case 2, censorship and withholding attacks can cause that some state data items are updated while others are not updated, which compromises the atomicity property. To counter the two attacks, we propose an attacks-resolution mechanism which consists of S_{SDM}, a state-data migration scheme, and S_{MUE}, a malicious user nodes elimination scheme. S_{SDM} enables a user node U_x migrate its state data item SD_x in a censorship-prone blockchain BC_i into other censorship-resilient BCs (such as BC_j). Different to the state-of-the-art "lock (or burn)-to-claim" schemes that sacrifice the independence property of involved BCs during data migration processes, Π_{SDM} achieves data migration while preserving the independence property of involved BCs through our "transfer and in place data update" design. On the other hand, S_{MUE} eliminates the user nodes who deliberately withhold their native transactions in Phase 1 to make Case 2 happen. Moreover, S_{MUE} determines the set of participators in the CCDI process at the end of Phase 2, allowing Π_{CCDI} to update honest participators' state data items without sacrificing its atomicity property. The details of the two protocols are presented as follows.

State-data migration scheme S_{SDM}. The user node U_x executes (5) to lock its state data item SD_x. At the same time, U_x creates an account in the blockchain BC_j (denoted as PK_x^j) and generates a newly initialized state data item $\overline{SD_x}$, which is SD_x's initial version. U_x then executes (5) to lock $\overline{SD_x}$. Once SD_x and $\overline{SD_x}$ are locked in BC_i and BC_j, respectively, U_x generates two cross-chain transactions according to (6) and sends them to the BoB. If the two transactions are valid, the M nodes of the BoB collectively produce the threshold signature TS_i for the locked SD_x and TS_j for the locked $\overline{SD_x}$. Finally, the $f+1$ replay nodes in the i-th group sends the transaction defined in (7) to transit SD_x to $\overline{SD_x}$ in BC_i. In the meantime, the $f+1$ replay nodes in the j-th group sends the transaction defined in (7) to transit $\overline{SD_x}$ to SD_x in BC_j.

Supported by the scheme S_{SDM}, all user nodes can migrate their state data items to other censorship-resilient BCs, thereby mitigating the negative impacts of censorship attacks.

Malicious user nodes elimination scheme S_{MUE}. S_{MUE} further eliminates the user nodes who make Case 2 happen through withholding attacks. First, a deposit mechanism is adopted, which enforces that every user node needs to deposit a certain number of tokens in the BoB before registering to participating in a CCDI process $\mathcal{P}_{CCDI}$. If a user node does not fulfill the deposit operations before the state of $\mathcal{P}_{CCDI}$, the authority of the user in $\mathcal{P}_{CCDI}$ is canceled. Second, if Case 2 happens, the BoB identifies the user nodes who fail to store their state data items in the smart contract $SC_{\mathcal{F}}$ before epoch $e_B + \kappa \times Max(\kappa_i)$. The accounts of the identified user nodes are then removed from the X registered user nodes, and the deposits of the identified user nodes are forfeited. Lastly, denoting the identified number of user nodes be $X - Y$, the remaining Y user nodes are authorized for the CCDI process $\mathcal{P}_{CCDI}$.

Under the above design of the scheme S_{MUE}, the final authorized participators in $\mathcal{P}_{CCDI}$ can only be determined at the end of Phase 2. In this way, more user nodes are allowed to participate in $\mathcal{P}_{CCDI}$ at the beginning of $\mathcal{P}_{CCDI}$, that aligns with complex cross-chain interaction applications in real-world scenarios. However, malicious user nodes can be identified and eliminated, guaranteeing the completion of the CCDI service for the remaining user nodes.

F. Security Analysis

This section is devoted to analyzing the security of the proposed CCDI protocol. The analysis is represented by the following theorems:

Theorem 4. *For the CCDI process $\mathcal{P}_{CCDI}$ defined in Π_{CCDI}, the independence property of each participating BC_i ($i \in [N]$) in the BoBMBS is preserved.*

Proof. In $\mathcal{P}_{CCDI}$, two types of native transactions are executed in BC_i (i.e., $Trans_{Ux}$ and $Trans_{\boldsymbol{S}_i}/Trans^*_{Rx}$):

(i) Executing $Trans_{Ux}$ involves checking the ownership of SD_x. As SD_x is the state data item of BC_i, the execution of this operation does not need the external state data.

(ii) The execution of $Trans_{\boldsymbol{S}_i}/Trans^*_{Rx}$ involves the two checking operations and the "in place data update" operation. The two checking operations require to use the state data PK and $\boldsymbol{S}_{x_c}$. To execute the "in place data update" operation, the state data, i.e., the locked SD_x, is required. Apparently, PK, $\boldsymbol{S}_{x_c}$, and the locked SD_x are the state data of SC_{BG}. Thus, the execution of $Trans_{\boldsymbol{S}_i}/Trans^*_{Rx}$ does not require any external state data except for that of BC_i, and proof completes. □

Theorem 5. *Under the corruption and network models defined in Section III-A, if every external BC_i ($i \in [N]$) and the BoB hold the safeness and liveness properties, by combing the attacks resolution mechanism, Π_{CCDI} will hold the atomicity, isolation, durability properties.*

Proof. To demonstrate the atomicity property, we analyze the state transitions of $\{SD_1, \dots, SD_X\}$ through different phases:

At Phase 2, with the support of the $f + 1$ relay nodes, as long as a state data item (e.g., SD_x) is immutably stored in corresponding blockchain (e.g., BC_i) before epoch $e_B + \kappa \times Max(\kappa_i)$, at least one of the $f + 1$ relay nodes will send the corresponding cross-chain transaction (e.g., CCT_{Ux}) to the BoB. Therefore, there are three possible situations. $\boldsymbol{S}_1$ (i.e., Case 1): X state data items are stored in $SC_{\mathcal{F}}$ before epoch $e_B + \kappa \times Max(\kappa_i)$. $\boldsymbol{S}_2$ (i.e., Case 2): only Y ($1 < Y < X$) state data items meet the deadline. $\boldsymbol{S}_3$: only Y ($1 \geq Y$) state data items meet the deadline.

At Phase 3, with the support of the $f + 1$ selected relay nodes, at least one honest relay node will send valid native transactions to the blockchains for the updates of locked state data in Phase 1. Thus, there are three possible situations:

$\boldsymbol{S}_4$: all the honest nodes' state data items are eventually updated in corresponding blockchains.

$\boldsymbol{S}_5$: some of honest users' state data items are updated to their new state versions, while the other honest users' state data items are just unlocked and unchanged.

$\boldsymbol{S}_6$: all state data items are unlocked and unchanged.

With the scheme S_{SDM}, every user node can move to the censorship-resilient blockchains. Therefore, honest user nodes can store their state data items in the smart contract $SC_{\mathcal{F}}$ before epoch $e_B + \kappa \times Max(\kappa_i)$ for sure, which can potentially eliminate the situation of $\boldsymbol{S}_6$. Moreover, the malicious user nodes who withhold their native transactions in Phase 1 can be identified and eliminated by the scheme S_{MUE}, and the pre-defined X user nodes for the CCDI process can be updated to the Y user nodes. As a result, the remaining honest users must be a part of the Y user nodes, making $\boldsymbol{S}_5$ unable to happen. Thus, only $\boldsymbol{S}_4$ can happen and Π_{CCDI} holds the atomicity property.

Regarding the isolation property, in Phase 1 and Phase 3, each involves the concurrent execution of X native transactions. Phase 2 involves the concurrent execution of X cross-chain transactions.

In Phase 1, the native transactions are associated with the operations of ownership checking for state data items and locking state data items. For the native transaction $Trans_{Ux}$, checking and locking U_x's state data SD_x only needs to check U_x's account information, that does not interfere with the execution of $Trans_{Uy}$ ($x \neq y$).

In Phase 2, the execution of the cross-chain transactions involves the two operations of checking time requirement and checking the validity of Merkle proofs. Firstly, checking the epoch index requirement is independent operation, as the epoch index used for checking will not be changed even if an epoch checking operation is conducted. Secondly, checking CCT_{Ux}'s Merkle proof does not rely on the information from CCT_{Uy}'s Merkle proof, and vice versa. As a result, the execution of CCT_{Ux} does not interfere with the execution of CCT_{Uy}.

In Phase 3, the execution of $Trans_{\boldsymbol{S}_i}/Trans^*_{Rx}$ is associated with checking the validity of the received threshold signature, checking key value pairs, and performing "in place data update" operations. Firstly, checking the validity of the threshold signature TS or the key value pairs is an independent operation and cannot change any state after the checking operation. Secondly, updating SD_x does not interfere with burning SD_y, as they are different state data items. As a result, the execution of $Trans_{\boldsymbol{S}_i}$

or $Trans^*_{Rx}$ does not interfere with the execution of $Trans_{s_j}$ with $i \neq j$.

As for the durability property, Π_{CCDI} involves three orderly executed transactions: the cross-chain transaction CCT_{Ux} can only be executed after the native transaction $Trans_{Ux}$ is stored in BC_i; and the native transaction $Trans_{s_i}/Trans^*_{Rx}$ can only be executed after CCT_{Ux} is stored in the BoB. With the support of the Merkle proof scheme, if $Trans_{Ux}$ is not stored in BC_i, its valid Merkle proof cannot be generated. As a result, the BoB will not execute CCT_{Ux}. Supported by the threshold signature scheme, if CCT_{Ux} is not stored in the BoB, a valid threshold signature cannot be generated. As a result, BC_i will not execute $Trans_{s_i}/Trans^*_{Rx}$. □

VI. Simulation

A. Simulation Setup

For the simulation tool, Multiple Ethereum BCs (including the external BCs and the BoB) are created by Ganache [36]. We use Solidity scripts to code the smart contracts, which are deployed and tested in Remix-IDE Ethereum [37]. Merkle tree and Merkle proofs are written by Python. The complete process to conduct the simulation are well-presented in [45].

For the simulated parameters, the block generation time of the Ethereum BC, the size of a block header, a block and a normal token transfer transaction (without considering the field of a transaction's signature with 48 bytes) are set to be 15 seconds, 510 bytes, 8×10^4 bytes and 50 bytes, respectively. The above settings align with the setting of the real-world Ethereum BC [46]. Additionally, we use SHA-256 hash function (32 bytes for each hash value) and the Merkle proof depth is fixed at 5. The threshold signature scheme is instantiated over the BLS12-381 pairing-friendly elliptic curve. The security parameter κ_i for BC_i ($\forall i \in [N]$) is set to be k.

The system is tested subjected to four metrics:

(i) *Communication overhead* $Comm_o$ – the size of transactions (in bytes) required for executing a CCDI process.

(ii) *Computation overhead* $Comp_o$ – the computation (in gas) required for executing a CCDI process.

(iii) *Memory overhead* M_o – the memory (in bytes) required for executing a CCDI process.

(iv) *Latency* L_o – the time slot (in seconds) between the beginning of a CCDI process and the state data items of honest user nodes involved in the CCDI process all being updated.

B. Evaluation of Π_{CCTE} without Attacks

We test the execution of a CCTE process with the proposed Π_{CCTE} without considering attacks. The total communication is 330 bytes, in which a cross-chain transaction with 230 bytes and two native transactions, each with 50 bytes, are included. The gas cost of the smart contracts is reported in Table III. The memory cost is k×*510*×N bytes, all undertaken by the BoB. When the external BCs use deterministic protocols, the latency of the CCTE process is 30 seconds (15 seconds are spent on the BoB and 15 seconds are on the BCs); when the protocols are probabilistic, the latency is 30+15 ×k seconds.

We compare the proposed Π_{CCTE} and two existing CCTE

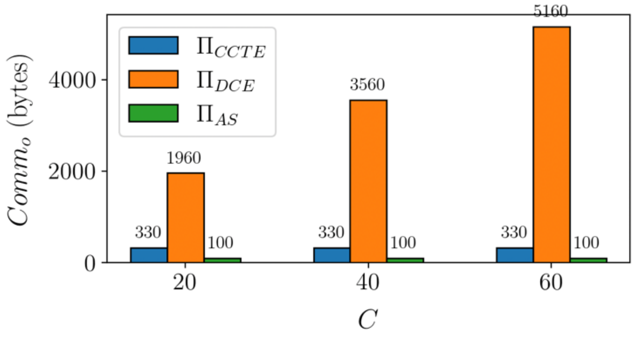

Fig.4 Communication cost comparison in CCTE. Both Π_{CCTE} and Π_{AS} does not introduce the variance C.

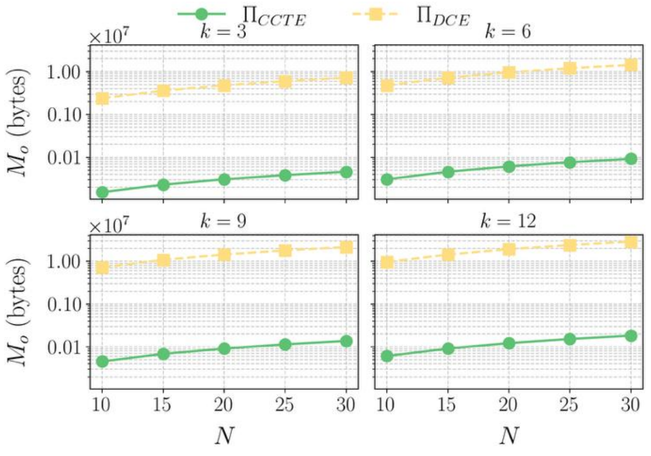

Fig.5 Memory cost comparison in CCTE under varying external BC numbers N and k.

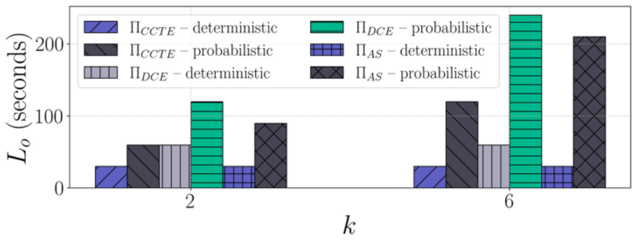

Fig.6 Latency cost in CCTE. "deterministic" and "probabilistic" denote the consensus types (see Section II-B).

schemes without considering attacks: a cryptocurrency exchange protocol (denoted as Π_{DCE}) [3] and an adaptor-signature-based protocol (denoted as Π_{AS}) [10]. Fig. 4 shows the comparison of the communication cost under the 3 protocols. It shows that although the Π_{AS} protocol is with the least communication cost (consuming 100 bytes), the proposed protocol is also efficient, requiring only 330 bytes to process a CCTE process. The communication cost of Π_{DCE} is higher than that of Π_{CCTE} due to its requirement for 1 cross-chain transaction, 3 token transfer transactions and 2C validation transactions, where C is the number of committee nodes.

As shown in Table III, Π_{CCTE} consumes 512,911 (470,911 for a CCT and 42,000 for two token transfers) gas to execute a CCTE process, which is less than Π_{DCE} (440,525+163,780*C)

TABLE III:
Smart Contracts Used in Π_{CCTE}.

Notation	Deployment gas cost	Execution gas cost
SC_D	2,901,508	-
SC_T	1,561,365	428,991
SC_{AMM}	7,610,008	-
SC_{MPV}	1,270,187	39,167

"-"means the gas cost of the smart contracts are covered by SC_T. The gas costs of SC_T and SC_{MPV} are caused by executing a valid cross-chain transaction and a valid proof transaction, respectively.

TABLE IV
Comparison on the Four Metrics of the Three Protocols in a CCTE Process under Attacks.

Types	$Comm_o$	$Comp_o$	M_o	L_o
Π_{CCTE}	980	732,251	510kN	45+15k
Π_{DCE}	410+40C	419,525+163,780C	8kN×10 4	45+15k
Π_{AS}	50	21,000	0	30+30k

A Proof-of-Work transaction in [3] is set to be 40 bytes in this paper.

TABLE V
Gas Consumptions of Basic Operations

Operation types (required protocols)	Gas cost	Price
Lock a state data item (MAP and PoS-Co)	186,443	\$3.7
Transfer a state data item (Π_{CCDI})	59,963	\$1.2
Verify a MP (Π_{CCDI} and PoS-Co)	39,167	\$0.8
ZK-SNARK (MAP)	250,000 [28]	\$5
Verify a time metric (all protocols)	20,533	\$0.4
Store a state data item (Π_{CCDI})	118,645	\$2.4
PvP battle games (all protocols)	63,779	\$1.3
Verify a TS (Π_{CCDI} and PoS-Co)	113,000[6]	\$2.3
Update a state data item (all protocols)	51,244	\$1
Mint a state data item (MAP and PoS-Co)	265,479	\$5.3

The price of ETH is considered as \$2,000 in this calculation.

gas [3], while higher than Π_{AS} (42,000 gas). Π_{CCTE} incurs higher gas costs than Π_{AS} because Π_{AS} executes CCTs off-chain, while Π_{CCTE} executes CCTs on-chain. Since CCTs run within the BoB, this overhead can be eliminated by implementing the BoB with a gas-free constructions [47, 48].

Since Π_{AS} does not need a BoB, Fig. 5 only shows the comparison of the memory cost between Π_{CCTE} and Π_{DCE}. Π_{CCTE} requires the BoB nodes to store a minimum number (i.e., k) of block headers per external BC to verify Merkle proofs. In contrast, Π_{DCE} requires the committee nodes of its BoB to store the latest k blocks of the N BCs, which results in higher memory cost especially when N continually increases. As a result, our protocol provides a scalable solution for the implementation of the BoB, as every consensus node in the BoB only caches kilobytes-level sized data for the verification tasks during CCTE processes.

Fig. 6 shows the latency cost comparison under the 3 protocols. Π_{CCTE} uses only 2 phases to complete the CCTE process, which is the same with that of Π_{AS}. Moreover, when the BCs adopt probabilistic consensus protocols and $k = 2$, our protocol Π_{CCTE} reduces its latency by 52% compared to Π_{AS}. This is due to the parallelized design of Π_{CCTE}, enabling the token transfer transactions being concurrently executed in Phase 2. As a result, this parallelized design not only improves protocol latency but also eliminates the censorship attack surface that is originated from sequential token transfers in [3] and [10].

C. Evaluation of Π_{CCTE} under Attacks

Table IV summarizes the results of the four key metrics for the 3 protocols under the censorship and withholding attacks. In the worst-case (both sender and receiver malicious), Π_{CCTE} incurs 980 bytes per CCTE operation (including two extra 210-byte proof transactions plus one extra 230-byte cross-chain transaction CCT_V). Consequently, its total gas consumption rises to 732,251 (219,340 gas attributable to those three transactions) and its latency increases by one additional block interval (from 30+15k seconds to 45+15k seconds). Its memory overhead, however, remains unchanged.

Although extra cost is introduced, Π_{CCTE} ensures its atomicity property and completes the two token transfers. In contrast, Π_{AS} fails to hold the atomicity property under the attacks (see Section VII-A), while Π_{DCE} rolls back all operations in the CCTE process. Thus, our protocol is more robust and practical under the adversarial scenarios.

D. Evaluation of Π_{CCDI} without Attacks

By implementing battle games (see Section V) in $SC_{\mathcal{F}}$ to act as the CCDI application, we compare the proposed Π_{CCDI} with other 2 protocols named as MAP [28] and PoS-Co [26]. For simplicity, we consider that each BC has $\lfloor X/N \rfloor$ state data items participating in a CCDI process. The following results are derived without considering attacks. Fig. 7 compares the communication cost of the 3 protocols. It can be seen that our proposed protocol Π_{CCDI} reduces communication cost by at least 35% compared to the two protocols. This is because that in Π_{CCDI}, only $2X$ transactions (see (5) and (6)) and N native transactions (see (7)) are required for updating X state data items. In contrast, PoS-Co requires $3X$+1 transactions to update X state data items. MAP's performance in communication cost is worst, as it requires $5X$ transactions to update X state data items.

Table V shows the gas consumption of the operations involved in the 3 protocols. Building upon Table V, Fig. 8 shows the comparison results of gas cost under the 3 protocols. It can be observed that Π_{CCDI} reduces gas consumption by at least 41% compared to the two protocols when $X = 120$, $N = 40$. Π_{CCDI}'s gas efficiency stems from three key design choices. First, in Phase 2, Π_{CCDI} records only each state data item's metadata in $SC_{\mathcal{F}}$, rather than minting and storing a state data item on the BoB. Table V shows that metadata storage requires far less gas than creating new state data items. Second, in phase 3, a single threshold signature lets Π_{CCDI} update $\lfloor X/N \rfloor$ state

[6]https://ethresear.ch/t/bls-signatures-in-solidity/7919

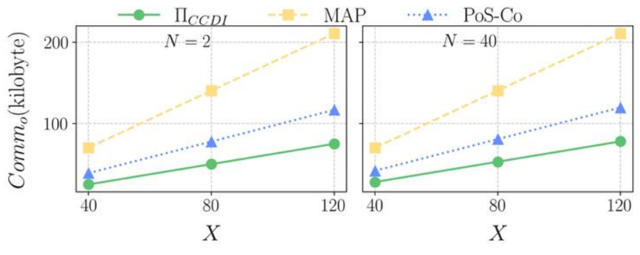

Fig.7 Communication comparison in CCDI under varying X and N.

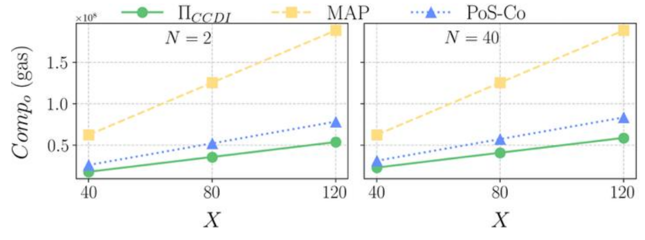

Fig.8 Gas comparison in CCDI under varying X and N (without considering the data storage costs of MAP and PoS-Co for validating proofs).

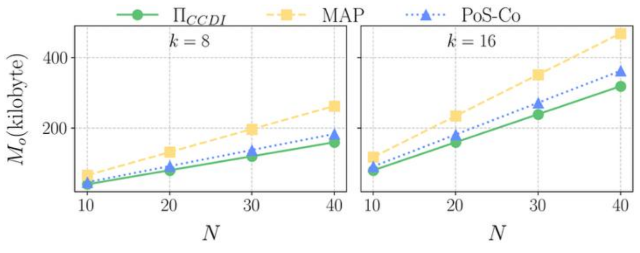

Fig.9 Memory comparison in CCDI under varying N and k (considering 40 validators in MAP's PoS consensus).

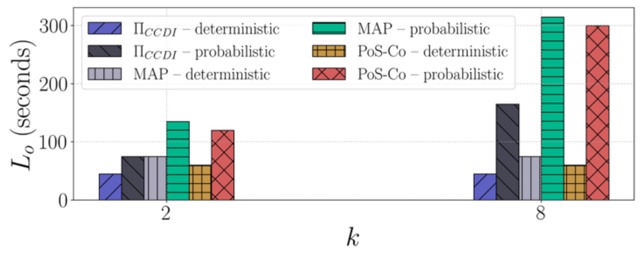

Fig.10 Latency comparison in CCDI. "deterministic" and "probabilistic" denote the consensus types.

data items in one BC at once, incurring a fixed gas cost of 21,000+113,000+51,244X/N units. In contrast, MAP and PoS-Co needs one transaction to update one state data item. Third, Π_{CCDI} only needs 3 phases to complete a CCDI process, while MAP and PoS-Co need 5 and 4 phases to complete a CCDI process, respectively. All the three merits make Π_{CCDI} \$2,593 and \$493 cheaper than MAP and PoS-Co, respectively, when $X = 120$ and $N = 40$.

Fig. 9 shows the comparison results of memory cost. The memory cost of Π_{CCDI} is $N \times k \times 510$ bytes, which is the same as Π_{CCTE}, as the two protocols ensure that every BC holds the independence property. Notably, the $N \times k \times 510$ bytes memory costs are only consumed by the BoB of the other two protocols. However, as MAP and PoS-Co do not hold the independence property, the BCs in their constructions consumes additional memory costs. In MAP, each BC needs to store the last k block headers of the BoB and a set of validators' public keys. PoS-Co is more efficient than MAP since each BC only requires updating a threshold public key within a certain number of epochs. However, for every CCDI application, each BC needs to store k latest root hashes of the BoB. As a result, Π_{CCDI} reduces its memory burden by at least 12% compared to the two protocols when $k = 16$, $N = 40$. According to [46], storing 1 kilobytes data consumes 640,000 gas, which makes Π_{CCTE} \$1,922 and \$560 cheaper than MAP and PoS-Co, when $N = 40$, $k = 16$.

Fig. 10 shows the comparison result of latency cost. Since Π_{CCDI} only consumes 3 phases to complete a CCDI process (see Section V), while MAP and PoS-Co require 5 and 4 phases, respectively, Π_{CCDI} reduces its latency by at least 33% compared to other two protocols. The efficiency of Π_{CCDI} is due to the "transfer and in place data update" design, which enables direct updates on multiple locked state data items using a single transaction within a single phase. By reducing latency, Π_{CCDI} narrows adversaries' window for mounting attacks, thereby enhancing system's robustness.

TABLE VI
Comparison on the Four Metrics of the Three Protocols in a CCDI Process Under Attacks

Types	$Comm_o$	$Comp_o$	M_o	L_o
Π_{CCDI}	640Y +80N	416,032Y+63,779(Y-1)+134,000N	510kN	45+15k
MAP	1824Y	1,543,166Y	510kN+(1,444+150k)N	>75+30k
PoS-Co	996Y +80N	621,500Y+134,000N	574kN+96N	>60+30k

In the CCDI process, we consider $X - Y$ malicious user nodes who can launch withholding attacks.

E. Evaluation of Π_{CCDI} under Attacks

We evaluate the 3 protocols under both censorship and withholding attacks. Table VI summarizes the four key metrics for each protocol. As shown, Π_{CCDI} outperforms both MAP and PoS-Co across every metric. This efficiency arises from two core mechanisms. First, the state-data migration scheme (S_{SDM}) allows honest user nodes to mitigate their state data items to censorship-resilient BCs, completely neutralizing censorship attacks; by contrast, MAP and PoS-Co must resort to timeout-based mechanisms to roll back an entire CCDI process for the preservation of their atomicity property. Second, the malicious user nodes elimination scheme (S_{MUE}) eliminates malicious user nodes and allows the remaining user nodes continue the current CCDI process. Consequently, even if these malicious user nodes launch withholding attacks, the state data items of the remaining honest user nodes are still updated, unlike MAP and PoS-Co that roll back all operations involved in the CCDI process.

VII. Related Work

This section provides a review for the CCTE and CCDI protocols that are reported in the literature or implemented in real-world applications.

A. Protocols for Cross-Chain Token Exchange

Time-lock-based protocols are actively studied for facilitating CCTE, with the Hash-Time-Lock Contract (HTLC) [5] being a pioneered model. Subsequent research is conducted to improve the performance of HTLC. [8] improves HTLC's latency from $O(|V|\Delta)$ to $O(\Delta)$, where $|V|$ is the number of parties involved in HTLC. [9] replaces time locks with attribute-verifiable timed commitments, eliminating the need for specific time parameters (e.g., Δ). Recently, adaptor signature-enabled lock schemes (ASLSs) are widely investigated to improve the time and on-chain computation efficiency of HTLCs [7, 10]. ASLSs divide a CCTE process into three phases. In the first phase, CCTE participators generate the token transfer transactions and commit pre-adaptor signatures off-chains. In the second phase, a participator first makes its adaptor signature publicly available and finish its token transfer transaction on-chain. In the third phase, the subsequent participators obtain secrets from the adaptor signature to fulfill their own token transfer transactions on-chain. As a result, only token transfer transactions are executed on-chain, eliminating the need of on-chain locking operations.

Despite the improvements achieved by ASLSs, the protocols are still prone to censorship attacks. Taking two parties' CCTE as an example, once Alice finishes its token transfer transaction in BC$_2$, its secret will be exposed to Bob which can use this secret to finish its token transfer transactions in BC$_1$. However, Alice can spend its account balance at the time when it submits its token transfer transactions in BC$_2$. If Alice can collude BC$_1$ to delay the token transfer transaction of Bob but to accelerate the execution of its own token transfer transaction, Alice can spend out its token balance in BC$_1$, making Bob's token transfer transaction unable to be executed.

To defend against censorship attacks, [3] proposes a Decentralized Cryptocurrency Exchange protocol Π_{DCE}. Π_{DCE} introduces relays who need to deposit certain amount of tokens in a BoB, and defines a CCTE process including 4 phases: (i) a sender sends a CCTE request to the BoB; (ii) the sender transfers coin$_1$ to an relay R$_1$ in BC$_1$; (iii) if the transfer in (ii) is confirmed, an relay R$_2$ transfers the equivalent amount of coin$_2$ to a receiver at BC$_2$; (iv) if the transfer in (iii) is valid, R$_1$ transfers the equivalent amount of coin$_3$ to R$_2$ in the TP. While the relays and the deposit mechanism can effectively counter censorship attacks, Π_{DCE} is inefficient as it needs additional mechanisms (e.g., Proof-of-Work adopt in [3]) to guarantee the trustworthy of relays. Moreover, under withholding attacks, Π_{DCE} can only roll back all operations involved in the CCTE process for the guarantee of the atomicity property, which provides an attack surface to deny CCTE services.

Notary-based schemes [4, 21], sidechains [24-26] and Oracle-based schemes [12, 14] can also support CCTE. However, these schemes are computationally inefficient and time-consuming when applied to CCTE. Although existing off-chain state-channel schemes [50, 51] can substantially boost throughput by enabling multiple users to concurrently exchange tokens off-chain without the need of the on-chain consensus, they still compromise the independence of the blockchains involved in a CCTE process. Developing techniques to retain each chain's independence in state-channel deployments therefore merits further investigation.

B. Protocols for Cross-Chain Data Interoperability

CCDI applications often involve complex state data update logics, making BoBs play an indispensable role in the protocols facilitating cross-chain data interoperability.

Some protocols [12-20] (e.g., IBC protocol) enable CCDI through direct message verification among BCs, with BoBs handling message transmission and data format translation. In these protocols, every BC needs to store information of other BCs (e.g., their block headers) for the verification of received cross-chain messages, which compromises the BCs' independence and leads to $O(N^2)$ memory complexity for N participating BCs.

Other protocols [12, 14], e.g., LayerZero [12], use oracles and relay nodes as third parties to relay data. Under this design, each BC does not need to store other BCs information; instead, each BC uses the information provided by oracles and relay nodes to validate cross-chain messages. However, to guarantee the trustworthy of oracles and relay nodes, multiple nodes need to provide information to defend against false data injection attacks launched by Byzantine nodes, which in turns introduces significant communication cost.

To reduce significant memory and communication burdens of the above protocols [12-20], the protocols belonging the second paradigm (described in Section I-A) have been widely studied in academia [21-26, 28-30] and adopted in industry [27]. Among them, some protocols (such as MAP [28] and XCMP [27]) directly use a BoB to verify CCDI requests. In these protocols, a CCDI process comprises of 5 phases: (i) all CCDI requests are sent to the BoB; (ii) the BoB's statements for these requests are sent to a destination BC; (iii) the BC updates all state data items in the requests; (iv) the updated operations are validated in the BoB; and (v) all updated state data items are updated in their original BCs.

In side-chain based schemes [24-26] (e.g., PoS-Co), a CCDI process consists of 4 phases: (i) the state data items are locked in the BCs; (ii) each locked state data item is created in the sidechain and the CCDI function is executed in the sidechain to update these state data items; (iii) the root hash of the these updated state data items a generated, and a threshold signature is generated by signing the root hash; after that, the threshold signature and the root hash are sent to the BCs; and (iv) every user node sends its updated state data item with a Merkle proof to its BC for the update of its locked state data item.

A drawback of the protocols in the second paradigm is that each BC needs to continually store some information for the verification of BoB's statements and proofs. This in turns compromises the BCs' independence and leads to a memory complexity with the order of $O(N)$ for N participating BCs. Additionally, current protocols in the second paradigm are not optimal, as they need at least 4 phases to finish a CCDI process. Additional phases introduce more communication, computation and latency burdens, which in turns reduces their efficiency, hindering their applicability in real-world applications. Moreover, under withholding attacks, current CCDI protocols can only roll back all operations to safeguard their atomicity property, which also provide chances for adversaries to launch denial of service attacks on CCDI applications.

VIII. Conclusions and Future Work

This paper proposes a CCTE protocol and a CCDI protocol to support cross-chain data interoperability. The CCTE protocol is designed for cross-chain token exchange. Besides be capable of preserving the independence of participating BCs, the CCTE protocol supports parallel execution of cross-chain token transfer transactions, which leads to a significant latency reduction. Although the CCTE protocol incurs higher communication, computation, and memory costs compared to time-lock-based schemes [7, 10], the CCTE protocol can hold atomicity property and complete token exchanges even in the presence of withholding and censorship attacks.

As for the proposed CCDI protocol, by designing the threshold signature-based "transfer and in place data update" mechanism, the proposed CCDI protocol securely achieves the state data updates without compromising the independence property of the participating BCs. Furthermore, with the threshold-signature-based "in place data update" operations, the CCDI protocol only spends one phase (i.e., one transaction execution time) to complete all the state data update in the participating BCs, which incurs the least latency cost compared to the state-of-the-art protocols [12-30]. Moreover, the state-data migration scheme and the malicious user nodes elimination scheme further safeguard the CCDI protocol in the presence of withholding and censorship attacks.

Future research can be conducted in two directions. Firstly, as deposit mechanism is not user-friendly, it is worth investigating if there is a method that can obviate the need of the deposit mechanism used in the proposed CCTE protocol (in Section IV) while maintaining the protocol's efficiency. Secondly, from the authors' perspective, the communication cost of the proposed CCDI protocol (in Section V) is relatively high, and we aim to further reduce the communication cost by designing more efficient message sending mechanisms [52] and applying leader-based mechanisms [53].

References

[1] A. Augusto et al., "SoK: Security and privacy of blockchain interoperability," *in Proc. IEEE Symp. Secur. Privacy*, pp. 3840-3865, May 2024.

[2] A. Lohachab et al., "Towards interconnected BCs: A comprehensive review of the role of interoperability among disparate blockchains," *ACM Comput. Surveys*, vol. 54, no. 7, pp. 1-39, Jul. 2021.

[3] H. Tian et al., "Enabling cross-chain transactions: A decentralized cryptocurrency exchange protocol," *IEEE Trans. Inf. Forensics Secur*., vol. 16, pp. 3928-3941, Jul. 2021.

[4] T. Yu, F. Luo, and G. Ranzi, "PCCAE: A protocol for multi-party asset exchange among blockchains," *in Proc. ACM SIGCOMM Workshop Ze-ro Trust Archit. Next Gener. Commun*., Aug. 2024, pp. 13-18.

[5] Y. Xue and M. Herlihy, "Hedging against sore loser attacks in cross-chain transactions," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2021, pp. 155-164.

[6] M. Herlihy et al., "Cross-chain deals and adversarial commerce," *The VLDB J*., vol. 31, no. 6, pp. 1291-1309, Nov. 2022.

[7] S. A. Thyagarajan et al., "Universal atomic swaps: Secure exchange of coins across all blockchains," in *Proc. IEEE Symp. Secur. Privacy*, May. 2022, pp. 1299-1316.

[8] S. Imoto et al., "Atomic cross-chain swaps with improved space, time, and local time complexities," *Inf. Comput*., vol. 292, Apr. 2023. Art. no. 105039.

[9] Yacov Manevich et al., "Cross chain atomic swaps in the absence of time via attribute verifiable timed commitments," in *Proc. IEEE 7th Eur. Symp. Secur. Privacy*, Jun. 2022, pp. 606-625.

[10] S. You, A. Joshi, A. Kuehlkamp, and J. Nabrzyski, "A multi-party, multi-blockchain atomic swap protocol with universal adaptor secret," *arXiv preprint arXiv:2406.16822.* 2024.

[11] A. Zamyatin et al., "XCLAIM: Trustless, interoperable, cryptocurrency backed assets," in *Proc. IEEE Symp. Secur. Privacy*, May. 2019, pp. 193-210.

[12] *Layerzero*. Accessed: Jun. 2024. [Online]. Available: https://layerzero.gitbook.io/

[13] M. Westerkamp et al., "zkrelay: Facilitating sidechains using zksnark-based chain-relays," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops*, Sep. 2020, pp. 378-386.

[14] L. Breidenbach et al., "Chainlink 2.0: Next steps in the evolution of decentralized oracle networks," *Chainlink Labs*, vol. 1, pp. 1-136, 2021.

[15] P. Sheng et al., "TrustBoost: Boosting trust among interoperable blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur*., Nov. 2023, pp. 1571-1584.

[16] M. Westerkamp and A. Küpper, "Instant Function Calls using Synchronized Cross-Blockchain Smart Contracts," *IEEE Trans. Network Service Manag*., vol. 20, no. 3, pp. 2136-2150, Sep. 2023.

[17] *Axelar Team*. (2021) Axelar Network: Connecting Applications with blockchain Ecosystems.

[18] R. Belchior, A. Vasconcelos, M. Correia, and T. Hardjono, "Hermes: Fault-tolerant middleware for blockchain interoperability," *Future Gener. Comput. Syst*., vol. 129, pp. 236-251, Apr. 2022.

[19] P. Zhang et al., "Cross-chain digital asset system for secure trading and payment," *IEEE Trans. Comput. Soc. Syst*., vol. 11, no. 2, pp. 1654-1666, Apr. 2024.

[20] J. Kwon and E. Buchman. (2019) *Cosmos WhitePaper*. [Online]. Available: https://cosmos.network/whitepaper

[21] A. Xiong et al., "A notary group-based cross-chain mechanism," *Digit. Commun. Netw*., vol. 8, no. 6, pp. 1059-1067, Apr. 2022.

[22] Z. Liu et al., "Hyperservice: Interoperability and programmability across heterogeneous blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur*., Nov. 2019, pp. 549-566.

[23] Y. Liu et al., "Secure and scalable cross-domain data sharing in zero-trust cloud-edge-end environment based on sharding blockchain," *IEEE Trans. Dependable Secure Comput*., vol. 21, no. 4, pp. 2603-2618, Jul./Aug. 2023.

[24] L. Yin, J. Xu, and Q. Tang, "Sidechains with fast cross-chain transfers," *IEEE Trans. Dependable Secure Comput*., vol. 19, no. 6, pp. 3925-3940, Sep. 2021.

[25] P. Gazi et al., "Proof-of-stake sidechains," in *Proc. IEEE Symp. Secur. Privacy*, May 2019, pp. 139-156.

[26] L. Yin et al., "Sidechains with optimally succinct proof," *IEEE Trans. Dependable Secure Comput*., vol. 21, no. 4, pp. 3375-3389, Jul./Aug. 2024.

[27] *XCMP*. Accessed: Jun. 2023. [Online]. Available: https://wiki.polkadot.network/

[28] Y. Cao et al., "MAP the blockchain world: A trustless and scalable blockchain interoperability protocol for cross-chain applications," *in Proc. ACM Web Conf.*, May 2025, pp. 717-726.

[29] T. Xie et al., "zkBridge: Trustless cross-chain bridges made practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur*, Nov. 2022, pp. 3003-3017.

[30] Y. Chen, A. Asheralieva, and X. Wei, "AtomCI: A new system for the atomic cross-chain smart contract invocation spanning heterogeneous blockchains," *IEEE Trans. Netw. Sci. Eng*., vol. 11, no. 3, pp. 2782-2796, May/Jun. 2024.

[31] G. Wang et al., "Exploring blockchains interoperability: A systematic survey," *ACM Comput. Surveys*, vol. 55, no. 13s, pp. 1-38, Feb. 2023.

[32] F. Martinelli and N. Mushegian. (2019) *Balancer: A Non-custodial Portfolio Manager, Liquidity Provider, and Price Sensor.* [Online]. Available: https://balancer. finance/whitepaper
[33] B. Pillai, K. Biswas, Z. Hóu, and V. Muthukkumarasamy, "The burn-to-claim cross-blockchain asset transfer protocol," in *Proc. Int. Conf. Eng. Complex Comput. Syst.*, Oct. 2020, pp. 119-124.
[34] B. Pillai, K. Biswas, Z. Hóu, and V. Muthukkumarasamy, "Burn-to-claim: An asset transfer protocol for blockchain interoperability," *Comput. Netw*., vol. 200, Dec. 2021. Art. no. 108495.
[35] R. Bacho and J. Loss, "On the adaptive security of the threshold BLS signature scheme," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 193-207.
[36] *Ganache*. Accessed: Jun. 2024. [Online]. Available: https: //truf-flesuite.com/
[37] *Remix*. Accessed: Jun. 2024. [Online]. Available: https://re-mix.ethereum.org/
[38] S. Zhang and J.-H. Lee, "Analysis of the main consensus protocols of blockchain," *ICT Express*, vol. 6, no. 2, pp. 93-97, Aug. 2020.
[39] X. Wang, S. Duan, J. Clavin, and H. Zhang, "BFT in blockchains: From protocols to use cases," *ACM Comput. Surv*., vol. 54, no. 10s, pp. 1-37, Sep. 2022.
[40] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," *in Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.,* Apr. 2015, pp. 281-310.
[41] W. Zou, D. Lo, P. S. Kochhar, et al., "Smart contract development: Challenges and opportunities," *IEEE Trans. Softw. Eng*., vol. 47, no. 10, pp. 2084-2106, Sep. 2019.
[42] B. Xue, S. Deb, and S. Kannan, "BigDipper: A hyperscale BFT system with short term censorship resistance," *arXiv preprint arXiv:2307.10185*. 2023.
[43] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-ng: Fast asynchronous BFT consensus with throughput-oblivious latency," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 1187-1201.
[44] A. Wahrstätter, et al., "Blockchain censorship," in *Proc. ACM Web Conf.*, May 2024, pp. 1632-1643.
[45] *Github*. Accessed: Jun. 2024. [Online]. Available: https://github.com/BryantTY/Test-for-RC-system/
[46] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2014.
[47] J. Camenisch et al., "Internet computer consensus," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2022, pp. 81-91.
[48] E. Androulaki et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proc. EuroSys Conf*., Apr. 2018, pp. 1-15.
[49] C. Schinckus, "Proof-of-work based blockchain technology and Anthropocene: An undermined situation?," *Renewable Sustain. Energy Rev*., vol. 152, Dec. 2021. Art. no. 111682.
[50] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," Jan. 2016. [Online]. Available: https:// lightning.net-work/lightning-network-paper.pdf
[51] X. Luo, K. Xue, Q. Sun, and J. Lu, "CrossChannel: Efficient and scalable cross-chain transactions through cross-and-off-blockchain micropayment channel," *IEEE Trans. Dependable Secure Comput*., vol. 22, no. 1, pp. 649-663, Jan./Feb. 2025.
[52] S. Coretti, A. Kiayias, C. Moore, and A. Russell, "The Generals' Scuttlebutt: Byzantine-Resilient Gossip Protocols," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 595-608.
[53] I. Kaklamanis, L. Yang, and M. Alizadeh, "Poster: Coded broadcast for scalable leader-based BFT consensus," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 3375-3377.

Teng Yu received the bachelor's degree from the College of Electrical Engineering and Automation, China Three Gorges University, China, in 2017, and the master's degree from the School of Electrical Engineering and Automation, Wuhan University, China, in 2019. He is currently pursuing the Ph.D. degree with The University of Sydney, Sydney, Australia. His research interests include consensus protocols and cross-chain protocols for blockchain, privacy-preserving techniques, and their applications in smart grids and machine learning.

Fengji Luo (Senior Member, IEEE) received his bachelor and master degrees in Software Engineering from Chongqing University, China in 2006 and 2009. He received his Ph.D degree in Electrical Engineering from The University of Newcastle, Australia, 2014. Currently, he is a Senior Lecturer in the School of Civil Engineering, The University of Sydney, Australia. He held a UUKI Rutherford International Researcher position in Future Energy Institute, Brunel University London. His research interests include computational intelligence, energy demand side management, smart grid, and energy informatics. He has over 200 publications in these fields.

Gianluca Ranzi (Member, IEEE) was awarded a degree in Management and Production Engineering at the Politecnico di Milano, Italy, a BE (Hons 1) at the University of Wollongong, Australia, a degree in Civil Engineering at the Universita' Politecnica delle Marche, Italy, and a PhD at the University of New South Wales (Australia). He is currently an ARC Future Fellow, professor and Director of the Centre for Advanced Structural Engineering at the University of Sydney, Australia. His research interests include high performance building, building-to-grid technology, advanced metering infrastructure, demand side management, and computational intelligence and its applications in smart buildings.

Jianzhong Wu (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from Tianjin University, China, in 1999, 2002, and 2004, respectively. From 2004 to 2006, he was with Tianjin University, where he was an Associate Professor. From 2006 to 2008, he was a Research Fellow with the University of Manchester, Manchester, U.K. He is a Professor of Multi-Vector Energy Systems and the Head of the School of Engineering, Cardiff University, U.K. His research interests include integrated multi-energy infrastructure and smart grid. He is Co-Editor-in-Chief of Applied Energy and the Co-Director of U.K. Energy Research Centre and EPSRC Supergen Energy Networks Hub.