MPC+: Secure, Compatible and Efficient Off-Blockchain Multi-Node Payment Channel

Longxia Huang, Hao Lei, and Liangmin Wang, *Member, IEEE*

***Abstract*—Payment channels (PC) greatly improve blockchain scalability by allowing an unlimited number of off-chain transactions instead of committing every transaction to the blockchain. However, the payment channel structure is limited to only two nodes, which is inadequate in scenarios where one node frequently receives money from multiple nodes, such as Cafe. Recent proposals have extended the 2-node structure to a multi-node structure. Unfortunately, these approaches either require all participants to always be online, which is not realistic, or underestimate the efficiency problem of withdrawing funds, leading to higher fees and storage costs. What's worse, the payment scope in these proposals is limited to their respective structures and is not mutually compatible. In this paper, we propose MPC+, a smart contract that enables multiple nodes to engage in off-chain transactions. MPC+ is designed to cater to scenarios where one node frequently receives money from multiple nodes, such as Cafe, shops, and more. The nodes in MPC+ can conduct off-chain transactions internally or with nodes in existing payment channel networks externally, and they only need to be online during off-chain transactions. Furthermore, MPC+ enhances the withdrawal logic through a binding strategy, effectively reducing the number of on-chain transactions required for fund withdrawals from $O(n)$ to $O(1)$, and it works over any blockchain system that supports Turing-complete smart contracts. The security is formally proven under the Universal Composability framework, and it is specifically implemented on the Ethereum platform. The experimental results clearly demonstrate that MPC+ surpasses other designs in terms of fees and storage costs.**

***Index Terms*—Blockchain, scalability, compatibility, smart contract, multi-node payment channel.**

I. Introduction

POOR scalability has always been a serious problem that restricts the development of blockchain. The most prominent cryptocurrency, Bitcoin [1], can only process 7 on-chain transactions per second, while Ethereum [2] can process nearly 15 on-chain transactions per second. The fundamental reason for this limitation is that consensus requires the participation of the entire network. Some recent works [3], [4], [5], [6], [7] have proposed new consensus mechanisms that support more on-chain transactions, but they still fail to meet the performance requirements of large-scale applications. For instance, Visa cards process nearly 47,000 peak transactions per second [8]. Therefore, the scalability of blockchain is not only a hot research topic at the theoretical level but also a crucial factor in promoting the widespread application of cryptocurrencies.

The innovative solutions to improve the scalability of blockchain mainly rely on payment channels (PC) [9], [10], which allow both parties to engage in off-chain transactions by locking deposit funds, recording only the initial and final states on the blockchain. For example, when Alice frequently purchases coffee from a Cafe, she can lock in some funds as the initial balance (IB) and designate the Cafe as the payee node to create a PC contract through a single on-chain transaction. Subsequently, Alice and the Cafe can maintain the allocation of this fund and engage in immediate off-chain transactions without the need for mediation on the blockchain. Either Alice or the Cafe can withdraw funds back to their account offline. The withdrawal process primarily relies on the consensus between both parties or on one party's application and the other party's verification.

Payment Channel Network (PCN) is a network of payment channels that enables off-chain transactions between two nodes that do not have direct channels. It enhances the functionality of payment channels by enabling transactions across multiple channels. The security of funds in this scenario is ensured through the use of Hash Time Locked Contracts (HTLC). PCN has gained support from various blockchain systems, including Bitcoin's Lightning Network [10] and Ethereum's Raiden Network [11]. Both PC and PCN greatly improve blockchain scalability through their instant and compatibility properties. Instant refers to the fact that funds in one PC only need to be maintained by the two parties involved, rather than the entire network. Compatibility enables different PCs to collaborate using the HTLC mechanism, facilitating off-chain payments between the two parties without requiring a direct PC.

Research on PC and PCN has primarily focused on either the transaction logic in PC [12], [13], [14], [15] or the payment routing protocols in PCN [16], [17], [18], [19], [20]. However, these studies have overlooked the structural limitation of a PC, which can only accommodate two nodes. This limitation poses a challenge for nodes like Cafe, which frequently receive funds

Manuscript received 11 October 2023; revised 29 January 2024; accepted 12 March 2024. Date of publication 20 March 2024; date of current version 12 July 2024. This work was supported by the National Natural Science Foundation of China (No. 62102168), the Leading-edge Technology Program of Jiangsu Natural Science Foundation (No. BK20202001), and the Postdoctoral Science Foundation of Jiangsu Province (No.2021K596C). The associate editor coordinating the review of this article and approving it for publication was A. Veneris. *(Corresponding author: Liangmin Wang.)*

Longxia Huang is with the School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China.

Hao Lei and Liangmin Wang are with the School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China (e-mail: liangmin@seu.edu.cn).

Digital Object Identifier 10.1109/TNSM.2024.3379475

TABLE I
COMPARISON OF OFF-CHAIN SOLUTIONS

Protocol	Node Number	Leader	Compatibility[2]	Off-Chain Tx Consensus Range	Msg Number for m Off-Chain Tx	Off-Chain Tx Confirmation Time		On-Chain Tx for n Funds Withdrawal
						optimistic	pessimistic	
PC [9], [10]	2	✗	✓	Payer & Payee	$O(m)$	$O(1)$	$O(1)$ / 2Δ	$O(n)$
Garou [24]	n[1]	✓	✗	All Participants[3]	$O(n)$+$O(m)$	$O(1)$	2Δ	$O(n)$
Magma [25]	n	✓	✗	All Participants	$O(n)$+$O(m)$	$O(1)$	Δ	$O(n)$
MPC [37]	n	✓	✓	Payer & Payee	$O(m)$	$O(1)$	$O(1)$ / 2Δ	$O(n)$
MPC+	n	✓	✓	Payer & Payee	$O(m)$	$O(1)$	$O(1)$ / 2Δ[4]	$O(1)$

[1] n represents the number of nodes that this off-chain solution can accommodate, which is unlimited.
[2] Compatibility represents the solution is compatible with PC&PCN to accomplish the cross-structure payments.
[3] Any off-chain transaction requires the confirmations of all users, which means they need to stay online at all times.
[4] Here are two types of off-chain transactions in this solution, of which the green type means it is confirmed off-chain immediately, while the red type requires on-chain challenge and Δ denotes the time for an on-chain transaction to be confirmed.

from multiple nodes, as they need to create a large number of redundant PCs. This not only incurs significant expenses but also results in high storage costs.

To solve the above problems, some research has been proposed to extend the 2-node structure to the multi-node structure [21], [22], [23]. However, multi-node structures often need to address the following issues. Firstly, the solution needs to ensure node dynamics without resorting to channel closure and restart. Secondly, considering that a node may only be online during a transaction, it is crucial to ensure that all participants engage in authorization signatures and verify update correctness. Additionally, the update logic in these studies is incompatible with PCNs, and payment scope is constrained by the original channel. To exacerbate matters, if any uncooperative party goes offline or refuses to provide a valid signature, the off-chain update process will be impeded, necessitating on-chain settlement, which contradicts its instant property.

It is important to note that off-chain solutions can be divided into three phases: (1) on-chain creation, (2) off-chain updates, and (3) on-chain withdrawals. However, existing multi-party off-chain solutions primarily concentrate on enhancing the efficiency of Phase (1) and Phase (2), while underestimating the efficiency of Phase (3) funding. Generally, multiple funds are held in these multi-party channels, resulting in a linear increase in the number of on-chain transactions for withdrawals in existing multi-party off-chain solutions.

In the multi-party off-chain payment scenario, to achieve both high efficiency and flexibility [24] while maintaining instant transaction and compatibility, we propose a smart contract where multiple nodes can make off-chain transactions, called MPC+. Firstly, MPC+ is designed to efficiently handle scenarios where one node frequently receives money from multiple nodes, such as Cafe or a shop. Additionally, MPC+ supports user joining and leaving, providing flexibility in the system. Secondly, we adopt a similar off-chain update logic as in PC, ensuring compatibility with PCN and maintaining the advantages of instant transactions and compatibility. Finally, we introduce a Center node [25] to coordinate with the node that wants to withdraw funds, binding multiple on-chain transactions into a single on-chain transaction. This benefits each participant in the process and improves the scalability of the blockchain. It is important to note that if any operation within this combination request is incorrect, all operations within the request will be rolled back, ensuring security.

A. Our Contribution

We implement the MPC+ smart contract in Ethereum using the Turing-complete language Solidity [2]. Furthermore, the proposed approach can be applied to other blockchain systems that support Turing-complete smart contracts. In this paper, we make the following contributions.

- We propose MPC+, a smart contract with the similar off-chain update logic as in PC, ensuring compatibility with the existing PCN. It also enables multiple nodes to make instant off-chain transactions. MPC+ is specifically designed to address scenarios where one node frequently receives money from multiple nodes. For a detailed comparison of MPC+ with other off-chain solutions, please refer to TABLE I.
- To the best of our knowledge, we are the first to introduce a bind strategy within multi-party off-chain solutions. In MPC+, we support the withdrawal of multiple funds simultaneously, effectively binding multiple on-chain transactions into a single on-chain transaction. This approach significantly reduces the number of on-chain transactions required, reducing it from $O(n)$ to $O(1)$.
- We formalize the security properties of MPC+ using the Universal Composability (UC) framework. Furthermore, we implement MPC+ in Ethereum using the Turing-complete language Solidity to evaluate the fees and storage costs in various situations and scales. The experimental results demonstrate the efficiency of MPC+.

B. Paper Organization

The remainder of this paper is organized as follows. Section II introduces the related works, while Section III presents some preliminaries that MPC+ is related to. In Section IV, we provide an overview of MPC+. Following

that, Section V presents formal descriptions of MPC+ along with security definitions and analysis. In Section VI, we conduct tests and validate the performance of MPC+. Finally, Section VII concludes this paper.

II. Related Work

Blockchain is a distributed ledger maintained across the entire network, which is ensured by participants running a protocol called Proof of Work (PoW) to ensure consistency. With the rise of blockchain technology, various cryptocurrencies [26], [27], [28] with new functions have emerged, and research based on blockchain technology has become a hot topic. For example, Ethereum [2] has extended blockchain to support smart contracts; DispersedLedger [29] allows nodes to propose, order, and agree on transaction blocks without downloading complete content; Zcash [30] guarantees the privacy of transactions based on zero-knowledge-proof. However, the setting of a consensus across the entire network has resulted in almost all popular blockchains facing a common challenge of poor scalability.

A simple way to expand the performance of blockchain is to improve consensus mechanisms. For example, OmniLedger [3] divides the entire blockchain network into several shards and optimizes performance via parallel intra-shard transaction processing. Bitcoin-ng [4] suggests electing a leader and then having that leader publish transaction blocks, thereby improving the delay in confirming transactions by an order of magnitude compared to the optimization in Bitcoin. The enhanced Byzantine protocol Algorand [5] or the double-layer PBFT consensus [6] are also feasible. Qu. et al. [7] designed Joint Learning Proof (PoFL), an energy recovery consensus algorithm that reinvests the energy initially wasted on solving difficult but meaningless puzzles in PoW. These novel consensuses have improved the performance of blockchain. However, they still cannot meet the performance requirements of large-scale scenarios, as they still need a long time to reach a consensus on the entire network.

Instead of struggling with consensus mechanism design, many works have been done on PC, a well-known "off-chain" technique. To reduce the number of on-chain transactions, PnP [12] designs a balance planning service that provides scalability. It determines the amount of IB locked in the PC based on the estimated payment demands among given nodes. Revive [13] and Boros [31] allow nodes to rebalance their payment channel funds, while the virtual payment channel in Perun [14] avoids intermediaries participating in each individual payment. Several improvements have been made for payment efficiency. Spider [15] improves throughput by splitting payments into micro-unit payments and adding delay queues to forward them. Sprite [16] improves worst-case performance in cross-channel payments. Robustpay+ [17] optimizes payment procedures to avoid off-chain transaction failures in PCN. Mercan et al. [18] have improved the overall success rate of payments by balancing payment routes through adjusting inbound and outbound capacity. MPCN-RP [19] and MILPA-PCN [32] choose the most economical path to promote payments within the PCN, while Coinexpress [20] proposes a distributed routing mechanism. Additionally, some related works are focused on privacy protection, such as SilentWhispers [33], SpeedyMurmurs [34], AMHL [35], and SorTEE [36]. Although these works have improved the performance of off-chain methods, they overlook the limitation that a PC can only accommodate two nodes. For nodes like Cafe that frequently receive funds from multiple nodes, it is necessary to repeatedly create large PCs.

To address the limitation of the number of nodes in PC, several studies [21], [22], [23], [24], [25] have been conducted on multi-party off-chain solutions. In the first three works [21], [22], [23], they primarily combine two PCs into one unit to enable multiple parties to engage in off-chain transactions. The combination process is initiated by the party that is simultaneously connected to the two PCs, resulting in a unit with three parties. Compared to a PC, the number of nodes in the combined unit only increases by 1. However, these approaches do not support parties dynamic. To achieve flexibility, Garou [24] and Magma [25] introduce a special role called the Center (also known as Leader or Operator) that handles the fund update process. In Magma, the Center is fixed upon creation, while in Garou, the Center is periodically elected from the parties. In both Garou and Magma, the fund update process is executed epoch-by-epoch. In each epoch, parties with payment needs to send transaction messages and corresponding signatures to the Center. The Center coordinates the parties in payment collection and broadcasts the new states. However, all parties are required to validate the new state by sending signatures to the Center, regardless of whether they have a payment need. This requirement for all parties to be online at all times lacks practical feasibility. Additionally, if there is an uncooperative party that is offline or refuses to validate the new state, the other nodes have to issue an on-chain challenge to obtain the desired messages. Furthermore, the update logic in these works differs from that in PC and PCN, limiting the scope of payments to their respective channels and making them incompatible with existing PC and PCN systems.

Recently, Lei et al. [37] have designed a multi-node payment channel that can accommodate an unlimited number of nodes for instant off-chain transactions. The update logic in this solution is similar to that in PC and PCN. However, when withdrawing funds, both MPC and existing multi-party off-chain solutions [24], [25] require a corresponding number of on-chain transactions equal to the number of funds. This results in high duplicate fees and storage costs. Furthermore, these solutions may even incur higher fees than PC, as additional operations are needed to ensure the security of other nodes' funds. Additionally, since each on-chain transaction is independent and the network may experience delays, it is possible that these transactions are not written into the same block simultaneously. As a result, some transactions may be included in the blockchain earlier, while others remain pending.

III. Preliminaries

A. Blockchain On-Chain Transaction

The blockchain is a distributed ledger consisting of blocks. Each block records a batch of on-chain transactions, while

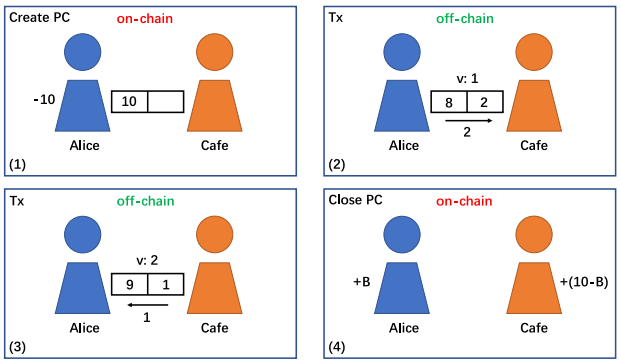


Fig. 1. A payment channel. Except for (1) and (4), which are on-chain transactions, other transactions in PC are off-chain. These off-chain transactions can be completed instantly and without any additional fees.

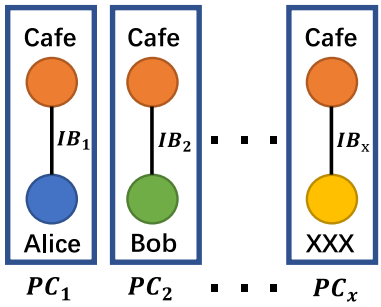


Fig. 2. The creation of numerous payment channels leads to repetitive fees and storage costs.

each on-chain transaction records some basic fields [38] that incur storage costs. Due to the need for network-wide consensus to prevent double-spending attacks, completing an on-chain transaction not only takes a long time but also requires an additional fee.

B. Smart Contract

A smart contract can be viewed as a program and ledger running on the blockchain, with a unique account address and balance. Once the contract code is deployed to the blockchain, the contract will continue to execute based on predefined conditions indefinitely. Nodes can interact with the contract according to predefined rules, such as sending coins to or withdrawing coins from the contract. Furthermore, each interaction requires one on-chain transaction.

C. Payment Channel: Two-Party

Payment channels can be implemented using smart contracts. If two nodes frequently engage in transactions, they can lock funds to create a PC. In Fig. 1, we consider a scenario where Alice frequently purchases coffee from Cafe. There are mainly three phases involved in this process.

- *Creation:* Alice locks 10 coins and designates Cafe as the payee node to create a PC (denote this PC with β).
- *Update:* Alice and Cafe can engage in off-chain transactions to maintain the balance of 10 coins. In an update, the initiator generates a message, denoted as *Msg* (β, v, x), where $v \in \mathbb{N}$ represents the version number that increases by 1 with each update, and x redistributes the funds. The initiator also includes a signature on *Msg* and the other party then responds. For example, if Alice needs to give Cafe 2 coins, she simply sends Cafe a message $(\beta, 1, 8)$ along with the corresponding signature. This message indicates that Alice's balance in β is 8 coins when $v = 1$. Cafe then replies with the signature on this message, completing the payment. Later, if Cafe needs to give Alice 1 coin, the message would be $(\beta, 2, 9)$. These off-chain transactions can be completed instantly and without any additional fees.
- *Withdrawal:* Either Cafe or Alice can initiate the withdrawal process to retrieve the 10 coins back to their respective accounts. The withdrawal process primarily depends on their mutual confirmations, where both parties need to agree on the withdrawal. Alternatively, one party can initiate the withdrawal by applying, and the other party can verify and confirm the withdrawal.

However, a single PC can only accommodate two nodes. Therefore, for nodes that frequently receive money from multiple nodes, a large number of PCs need to be created. This may lead to a significant increase in the amount of information related to these channels being added to the blockchain. As a consequence, the repetitive addition of channel information results in high fees and storage costs, as illustrated in Fig. 2.

IV. Overview of MPC+

In this section, we first introduce the model of MPC+, then describe the basic idea of MPC+ protocol.

A. MPC+ Model

1) MPC+ Components and Phases: As the MPC+ architecture shown in Fig. 3, there are mainly three types of entities: **MPC+ Smart contract**, **Center**, and **Node**.

- *MPC+ Smart contract:* The MPC+ smart contract, denoted as $\mathcal{M}$, is the code deployed on the blockchain during the channel construction process. It facilitates the transfer of coins between the contract and the parties' individual accounts by receiving and processing specified messages according to predefined rules. It is important to note that each MPC+ instance is assigned a unique address identifier, ensuring that every MPC+ operates independently from one another.
- *Center:* The Center, represented as $\mathcal{C}$, is the creator of MPC+ and is permanently limited to a single node.
- *Node:* The node actively participates in transactions with the Center and is assigned a unique index i, denoted as $\mathcal{N}_i$. $\mathcal{N}_i$ stores various information, such as its *address*, IB_i, and *state*. The state is represented in the format $(ver, bal, withdrawn, \mathcal{C}Apply, \mathcal{N}Apply, timestamp)$, and it undergoes changes whenever a withdrawal operation occurs.

As shown in Fig. 3, there are mainly four phases during the lifetime of MPC+: **Creation**, **Joining**, **Update**, and **Withdrawal**.

- *Creation:* The Center deploys the contract code onto the blockchain to create the MPC+.

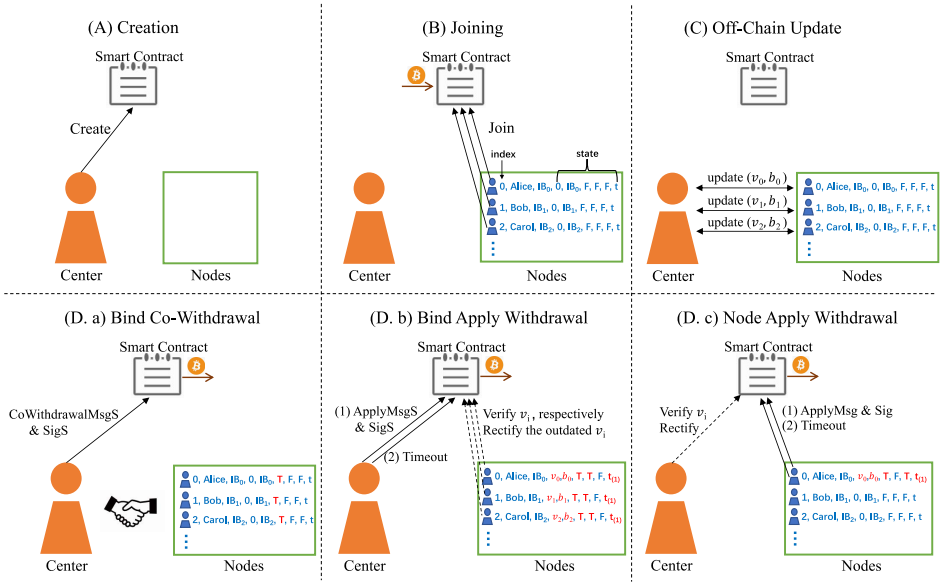


Fig. 3. Each solid line pointing to a contract in the MPC+ architecture represents an on-chain transaction, while each dotted line represents a potential on-chain transaction. The line between the Center and nodes represents the off-chain message. As shown in phase D (a, b, c), there are three primary methods to withdraw funds individually. The red portion of the status changes based on the withdrawal operation, and its symbols and descriptions can be found in TABLE II.

TABLE II
KEY NOTATIONS AND DESCRIPTIONS

Notation	Description
$\mathcal{L}$	The underlying blockchain
T	The withdrawal delay after application
$\mathcal{M}$	MPC+ smart contract
$\mathcal{C}$	Center, the creator of $\mathcal{M}$
$\mathcal{N}_i$	The i-th node joining $\mathcal{M}$
IB_i	$\mathcal{N}_i$'s initial balance, (belongs to $\mathcal{C}$ and $\mathcal{N}_i$)
$\mathcal{N}_i.state$	including the following six states
$\mathcal{N}_i.ver$	$\mathcal{N}_i$'s version number of off-chain tx
$\mathcal{N}_i.bal$	$\mathcal{N}_i$'s balance in version ver
$\mathcal{N}_i.withdrawn$	IB_i is withdrawn, initially $False$
$\mathcal{N}_i.\mathcal{C}Apply$	$\mathcal{C}$ applies to withdraw IB_i, initially $False$
$\mathcal{N}_i.\mathcal{N}Apply$	$\mathcal{N}_i$ applies to withdraw IB_i, initially $False$
$\mathcal{N}_i.timestamp$	Record the time of operation
$MsgS$	A set of messages
$SigS$	A set of signatures
$\xrightarrow{x}$	Transfer x coins on-chain

- *Joining:* Once a node joins MPC+ by locking a certain amount of funds, MPC+ assigns a new index to the node and stores the necessary information. The *index* assigned to each newly joined node is unique and increments by 1.
- *Update:* When an update occurs, it signifies a redistribution of funds for off-chain transactions. In MPC+, each node can directly engage in off-chain transactions with the Center or indirectly engage in off-chain transactions with other nodes through the assistance of the Center. Furthermore, each IB update is independent, meaning that each node only needs to remain online during the transaction. The update logic in MPC+ is highly similar to that of PC and PCN.
- *Withdrawal:* There are primarily three methods for withdrawing funds: (1) *Bind Co-Withdrawal*, (2) *Bind Apply Withdrawal* and (3) *Node Apply Withdrawal*. When a withdrawal request is received, the contract refunds the deposit funds of the respective users based on the most recent balance state.

2) Goals: We aim to achieve the following three goals.

Security: As MPC+ is a smart contract that locks multiple funds, the primary security objective is to safeguard the funds of honest parties [14], [25], whether they are on-chain or off-chain. In other words, the goal is to ensure that honest parties do not incur any financial losses.

Compatibility: The nodes in MPC+ can transact not only with nodes within MPC+, but also with nodes in the PCN, thereby facilitating payments across different structures.

Efficiency: The update process in MPC+ can be executed instantaneously. Additionally, MPC+ supports the simultaneous withdrawal of multiple funds through a single on-chain transaction, effectively consolidating multiple on-chain transactions into one.

B. Informal Description of MPC+

In this subsection, we provide a brief description of the operations of the MPC+ protocol in different phases, along with the contract functionality illustrated in Fig. 4.

1) Creation: The node that frequently receives money from multiple nodes is able to create an MPC+ through a single on-chain transaction and be assigned as the unique Center role indefinitely. As illustrated in Fig. 3(A), the left side represents the Center (e.g., Cafe), while the right side depicts a dynamic array *Nodes* that stores the information of all joined nodes. Initially, the number of nodes in the array is 0.

2) Joining: The nodes that frequently engage in transactions with the Center can individually lock a certain amount

We underline the "message" that invokes the contract as "message".

Creation

Upon $(\underline{\text{create}}, \mathcal{M}) \overset{\tau}{\hookleftarrow} P$, $P \overset{\delta}{\rightharpoonup} \mathcal{L}$, where δ is the fee for on-chain transaction, τ is the current round, $\overset{\tau}{\hookleftarrow} P$ means "receive message from P in round τ", set P as $\mathcal{C}$ and initiate an array Nodes.

Joining

Upon $(\underline{\text{join}}, \mathcal{M}, x) \overset{\tau}{\hookleftarrow} P$, $P \overset{x+\delta}{\rightharpoonup} \mathcal{L}$, $\mathcal{L} \overset{x}{\rightharpoonup} \mathcal{M}$, assign a new index i to represent P, and add $\mathcal{N}_i = \{addr : P, IB : x, state : (0, x, False, False, False, now)\}$ to Nodes.

The Withdrawal Execution of MPC+

Wait for the following messages:

(A) Bind Co-Withdrawal

Upon $(\underline{\text{bind-cowithdrawal}}, withdrawMsgS) \overset{\tau}{\hookleftarrow} \mathcal{C}$, $\mathcal{C} \overset{\delta}{\rightharpoonup} \mathcal{L}$, process each $withdrawMsg := (id, i, val, sig_i)$ orderly as follow.

1) If $val \notin [0, IB_i]$, or $\mathcal{N}_i.withdrawn = True$, or sig_i is unmatched, roll back and terminate this procedure.
2) Set $\mathcal{N}_i.withdrawn = True$, $\mathcal{M} \xrightarrow{val} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - val} \mathcal{C}$.

(B) Bind Apply Withdraw, and Node Respond

1) Upon $(\underline{\text{bind-apply-withdrawal}}, ApplyMsgS, SigS) \overset{\tau}{\hookleftarrow} \mathcal{C}$, $\mathcal{C} \overset{\delta}{\rightharpoonup} \mathcal{L}$, process every $ApplyMsg := (i, ver, val)$ and sig_i orderly as follow.
   a) If $\mathcal{N}_i.\mathcal{C}Apply = True$, or $val \notin [0, IB_i]$, or sig_i is unmatched, roll back and terminate.
   b) If $\mathcal{N}_i.\mathcal{N}Apply = False$, set $\mathcal{N}_i.\mathcal{C}Apply = True$, $\mathcal{N}_i.ver = ver$, $\mathcal{N}_i.bal = val$.
2) Upon $(\underline{\text{node-agree}}, i) \xleftarrow{\tau_1 > \tau + \Delta} \mathcal{N}_i$, $\mathcal{N}_i \overset{\delta}{\rightharpoonup} \mathcal{L}$, terminate this procedure if $\mathcal{N}_i.\mathcal{C}Apply = False$, or $\mathcal{N}_i.withdrawn = True$, terminate this procedure. Otherwise, set $\mathcal{N}_i.withdrawn = True$, $\mathcal{M} \xrightarrow{\mathcal{N}_i.bal} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - \mathcal{N}_i.bal} \mathcal{C}$.
3) Upon $(\underline{\text{node-rectify}}, i, ver', val', sig_{\mathcal{C}}) \xleftarrow{\tau_2 \le \tau + \Delta + T} \mathcal{N}_i$, $\mathcal{N}_i \overset{\delta}{\rightharpoonup} \mathcal{L}$, terminate this procedure if $\mathcal{N}_i.\mathcal{C}Apply = False$, or $\mathcal{N}_i.withdrawn = True$, or $ver' \le \mathcal{N}_i.ver$, or $val' \notin [0, IB_i]$, or $sig_{\mathcal{C}}$ is unmatched. Otherwise, set $\mathcal{N}_i.withdrawn = True$, $\mathcal{M} \xrightarrow{val'} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - val'} \mathcal{C}$.

(C) Node Apply Withdraw, and Center Respond

1) Upon $(\underline{\text{node-apply}}, i, ver, val, sig_{\mathcal{C}}) \overset{\tau}{\hookleftarrow} \mathcal{N}_i$, $\mathcal{N}_i \overset{\delta}{\rightharpoonup} \mathcal{L}$, terminates this procedure if $\mathcal{N}_i.\mathcal{C}Apply = True$, or $\mathcal{N}_i.\mathcal{N}Apply = True$, or $val \notin [0, IB_i]$, or $sig_{\mathcal{C}}$ is unmatched (if ver is 0 then ignore the $sig_{\mathcal{C}}$). Otherwise, set $\mathcal{N}_i.\mathcal{N}Apply = True$, $\mathcal{N}_i.ver = ver$, $\mathcal{N}_i.bal = val$.
2) Upon $(\underline{\text{center-rectify}}, i, ver', val', sig_{\mathcal{C}}) \xleftarrow{\tau_1 \le \tau + \Delta + T} \mathcal{N}_i$, $\mathcal{N}_i \overset{\delta}{\rightharpoonup} \mathcal{L}$, terminate if $\mathcal{N}_i.\mathcal{N}Apply = False$, or $\mathcal{N}_i.withdrawn = True$, or $ver' \le \mathcal{N}_i.ver$, or $val' \notin [0, IB_i]$, or $sig_{\mathcal{N}_i}$ is unmatched. Otherwise, set $\mathcal{N}_i.withdrawn = True$, $\mathcal{M} \xrightarrow{val'} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - val'} \mathcal{C}$.

(D) Bind Timeout Withdrawal

Assume the funds are applied for withdrawal in round τ. Upon $(\underline{\text{bind-timeout-withdrawal}}, MsgS) \xleftarrow{\tau_1 > \tau + \Delta + T} \mathcal{C}$, $\mathcal{C} \overset{\delta}{\rightharpoonup} \mathcal{L}$, process every $Msg := (id, i)$ orderly as follow.

1) If $\mathcal{N}_i.\mathcal{C}Apply = False$ and $\mathcal{N}_i.\mathcal{N}Apply = False$, roll back and terminate this procedure.
2) If $\mathcal{N}_i.withdrawn = False$, set $\mathcal{N}_i.withdrawn = True$, $\mathcal{M} \xrightarrow{\mathcal{N}_i.bal} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - \mathcal{N}_i.bal} \mathcal{C}$.

Fig. 4. The contract functionality.

of funds through a single on-chain transaction in order to join MPC+.

As depicted in Fig. 3(B), three nodes send the join messages to MPC+ along with their respective deposited funds. The first node is assigned the index "0", and the index number gradually increases by 1. As shown in **Procedure 1**, MPC+ records essential information (as described in Section IV-A1) of each joined node to ensure the smooth execution of Update and Withdrawal. This information includes the node's address and IB. Additionally, MPC+ initializes the state of the joining node, which subsequently changes based on withdrawal operations for the corresponding fund.

3) Update: In MPC+, the Center has a dedicated PC for each joined node. Therefore, the network topology of MPC+ can be described as a star network topology, characterized by two main types of updates: direct update and indirect update.

Procedure 1: MPC+ Joining

Input: *JoinMsg* $(addr, x)$
$\mathcal{M}$ assigns a new index i;
$\mathcal{N}_i.addr \leftarrow addr$;
$IB_i \leftarrow x$;
$\mathcal{N}_i.state \leftarrow (0, x, False, False, False, now)$;
$\mathcal{N} \overset{x}{\rightharpoonup} \mathcal{M}$;
Output: *JoinInfo*

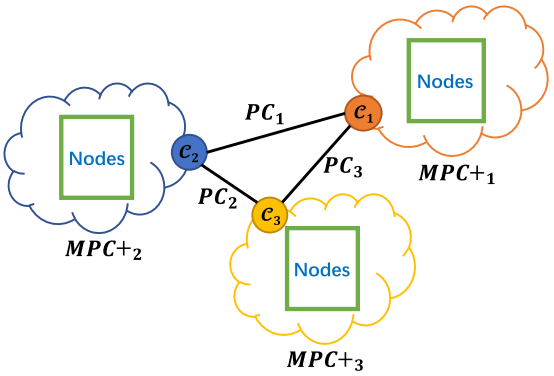


Fig. 5. The topology of the MPC+ & PCN network. The three nodes highlighted in different colors represent the Centers of each MPC+. These Centers establish connections with each other to form a PCN, enabling off-chain transactions and ensuring compatibility across nodes with different structures.

Direct Update: The Center can make off-chain transactions directly with each node, as each IB in MPC+ can be considered as a PC connecting the Center to the corresponding node. The update logic in MPC+ is quite similar to that in a PC (refer to Section III-C). However, since MPC+ involves multiple funds compared to the single fund in a PC, an additional field called *index* is required to specify the fund. The format of the Msg for direct updates in MPC+ is as

$$TxMsg = (\mathcal{M}_{id}, index_i, ver, bal),$$

where $\mathcal{M}_{id}$ and $index_i$ are used to specify the specific location of the fund, *ver* denotes the version number for redistribution, and *bal* is responsible for redistributing the fund between $\mathcal{C}$ and corresponding node $\mathcal{N}_i$. In general, the initiator (payer) of the update first signs the message *Msg* and sends the signature *Sig* to the responder (payee). The responder then verifies the correctness of the signature. If the signature passes the verification, the responder signs the message *Msg* and sends the corresponding signature *Sig* back to the initiator.

Indirect Update: In MPC+, nodes typically engage in transactions with multiple Centers rather than just one. To facilitate cross-channel payments between nodes, Centers can create multiple PCs with other Centers. It aligns with the concept of small-world theory [39], which suggests that it is easier to establish paths for cross-channel payments. In practice, Centers can charge incentive fees for facilitating cross-channel payments, making it advantageous for them to participate in such transactions. Fig. 5 illustrates the creation of PCs among Centers. Once these PCs are established, nodes can make payments to other Centers or nodes in different MPC+ networks using a combination of cross-channel and cross-MPC+ methods. For instance, let's consider the scenario where Alice, in MPC_1, needs to pay Dave, in MPC_2. The payment can be accomplished by following the path $(Alice \rightarrow \mathcal{C}_1 \rightarrow \mathcal{C}_2 \rightarrow Dave)$. The HTLC mechanism [10] binds the entire path into one atomic operation. The interconnected PCs form a small PCN, enabling indirect payments across different structures while maintaining compatibility between MPC+ and PCN.

Procedure 2: Bind Co-Withdrawal

Input: *CoWithdrawalMsgS* and *SigS*
if *sender is not* $\mathcal{C}$ **then**
  return;
foreach *Msg* $\langle \mathcal{M}, index_i, b_i \rangle$ *and* Sig_i **do**
  if $b_i \notin [0, IB_i]$, *or* $\mathcal{N}_i.withdrawn = True$, *or* Sig_i *is unmatched* **then**
    roll back and return;
  else
    $\mathcal{N}_i.withdrawn \leftarrow True$;
    $\mathcal{M} \xrightarrow{b_i} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - b_i} \mathcal{C}$;
Output: *WithdrawalInfo*

4) Withdrawal: No one can withdraw an IB in MPC+ except for the Center and the corresponding depositor. As the Center manages all IBs, it can handle them simultaneously through a single on-chain transaction, known as the *bind* operation. In MPC+, there are primarily three methods to withdraw IB back to an account, as outlined below.

a) Bind co-withdrawal: If both $\mathcal{C}$ and $\mathcal{N}_i$ wish to withdraw IB_i, they can cooperatively determine a *CoWithdrawalMsg*, which has the format $(\mathcal{M}, index_i, b_i)$. Here, b_i must satisfy $0 \leq b_i \leq IB_i$. If b_i does not meet this condition, MPC+ will ignore the message. The distribution of IB_i is then performed as follows.

$$[\, b_i \mapsto N_i, (IB_i - b_i) \mapsto \mathcal{C} \,]$$

In this formula, b_i belongs to $\mathcal{N}_i$, while the remaining amount $(IB_i - b_i)$ belongs to $\mathcal{C}$. Afterward, $\mathcal{N}_i$ signs this message and provides the signature to the Center.

To improve efficiency, the Center can negotiate with other nodes that wish to withdraw IBs and collect their *CoWithdrawalMsgS* along with the corresponding signatures. Subsequently, the Center can bind-withdraw these IBs by submitting all the *CoWithdrawal MsgS* and signatures *SigS* to MPC+ in a single on-chain transaction. The procedure for this bind withdrawal is outlined in **Procedure 2**. In **Procedure 2**, MPC+ first verifies whether the sender is the Center. It then proceeds to verify every *Msg* and *Sig* in order. If all verifications are successful, MPC+ sets the *withdrawn* state to *True* to prevent replay attacks [40]. It then transfers the coins according to *CoWithdrawalMsg*. However, if any information is found to be incorrect during the verification process, all operations will be rolled back.

An example of the *Bind Withdrawal* process is illustrated in Fig. 3 (D. a). The Center initiates negotiations with Alice, Bob, and Carol (indexed as 0, 1, and 2 respectively) and collects their signatures. Subsequently, the Center submits these *CoWithdrawalMsgS* and corresponding signatures *SigS* to MPC+ in a single on-chain transaction. If all the information is verified correctly, MPC+ transfers the funds that Alice, Bob, and Carol had locked to their respective accounts. Prior to the transfer, MPC+ sets the *withdrawn* state of the three indexes to *True* to prevent replay attacks.

Procedure 3: Bind Apply Withdrawal, and Respond

Input: *ApplyMsgS* and *SigS*
if *sender is not* $\mathcal{C}$ **then**
  return;
foreach *Msg* $\langle \mathcal{M}, index_i, v_i, b_i \rangle$ *and* Sig_i **do**
  if $b_i \notin [0, IB_i]$, *or* $\mathcal{N}_i.\mathcal{C}Apply$ = *True, or* Sig_i *is unmatched* **then**
    roll back and return;
  if $\mathcal{N}_i.\mathcal{N}Apply$ = *False* **then**
    $\mathcal{N}_i.ver \leftarrow v_i$;
    $\mathcal{N}_i.bal \leftarrow b_i$;
    $\mathcal{N}_i.\mathcal{C}Apply \leftarrow True$;
    $\mathcal{N}_i.timestamp \leftarrow now$;

Output: *ApplyInfo*
// The nodes responds, respectively.
// Case A: Node Agree
Input: *Msg* $(index_i)$
if *sender is not* $\mathcal{N}_i$, *or* $\mathcal{N}_i.\mathcal{C}Apply$ = *False, or* $\mathcal{N}_i.withdrawn$ = *True* **then**
  return;
else
  $\mathcal{N}_i.withdrawn \leftarrow$ *True*;
  $\mathcal{M} \xrightarrow{\mathcal{N}_i.bal} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - \mathcal{N}_i.bal} \mathcal{C}$;

Output: *WithdrawalInfo*
// Case B: Node Rectify
Input: *Msg* $(\mathcal{M}, index_i, ver', bal')$ and $Sig_{\mathcal{C}}$
if *sender is not* $\mathcal{N}_i$, *or* $\mathcal{N}_i.\mathcal{C}Apply$ = *False, or* $\mathcal{N}_i.withdrawn$ = *True, or* $Sig_{\mathcal{C}}$ *is unmatched, or* $bal' \notin [0, IB_i]$, *or* $ver' \leq \mathcal{N}_i.ver$ **then**
  return;
else
  $\mathcal{N}_i.withdrawn \leftarrow$ *True*;
  $\mathcal{M} \xrightarrow{\mathcal{N}_i.bal'} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - \mathcal{N}_i.bal'} \mathcal{C}$;

Output: *WithdrawalInfo*

b) Bind apply withdrawal: If the Center intends to withdraw funds back to its account but the corresponding nodes are uncooperative in providing confirmations through the first comanner, the Center can initiate an application by submitting an *ApplyMsg* along with the corresponding node's signature *Sig*. Otherwise, the funds will remain locked in MPC+ indefinitely. The format of the *ApplyMsg* is $(\mathcal{M}, index_i, v_i, b_i)$, which follows the same format as the *TxMsg*. This format indicates that the balance of $\mathcal{N}_i's$ is b_i (where $0 \leq b_i \leq IB_i$) when the version is v_i.

Similarly, the Center can apply for the withdrawal of multiple IBs at once through a single on-chain transaction, rather than multiple transactions. As shown in **Procedure 3**, MPC+ first verifies whether the sender is the Center. It then proceeds to check the correctness of each *ApplyMsg* and corresponding signature *Sig* in order. This includes verifying the correctness of b_i and the signature itself. It is important to note that each IB can only be applied for withdrawal once, either by the Center or the corresponding node. Multiple applications for the same IB are meaningless and can potentially confuse the verifier. If all the checks are correct, MPC+ updates the state of the corresponding index based on the Center's application, as depicted in Fig. 3 (D. b). However, if any information is found to be incorrect during the verification process, all operations will be rolled back.

Procedure 4: Bind Timeout Withdrawal

Input: *TimeoutMsgS*
if *sender is not* $\mathcal{C}$ **then**
  return;
foreach *Msg* $\langle index_i \rangle$ **do**
  if ($\mathcal{N}_i.\mathcal{C}Apply$ = *False and* $\mathcal{N}_i.\mathcal{N}Apply$ = *False), or* $now < \mathcal{N}_i.timestamp + T$ **then**
    roll back and return;
  if $\mathcal{N}_i.withdrawn$ = *False* **then**
    $\mathcal{N}_i.withdrawn \leftarrow$ *True*;
    $\mathcal{M} \xrightarrow{\mathcal{N}_i.bal} \mathcal{N}_i$, $\mathcal{M} \xrightarrow{IB_i - \mathcal{N}_i.bal} \mathcal{C}$;

Output: *WithdrawalInfo*

To prevent the Center from submitting expired balance information and potentially cheating funds, a delay duration T is introduced. This delay allows the nodes involved to verify the Center's application before any funds are transferred. For instance, in the Lightning network [10], T corresponds to the period of 1000 new blocks, which is approximately 7 days. During the delay period T, the nodes verify the correctness of the Center's application regarding their own IBs. Several situations may arise during this verification process as follow.

Case A: Node Agree. In this situation, when the Center submits the latest fund information, the node may agree with the Center's application and handle the withdrawal immediately through a single on-chain transaction. In this case, MPC+ sets the *withdrawn* state of the node to *True* to indicate that the withdrawal has been completed. Alternatively, the node can choose to wait for the Center to invoke **Procedure 4** after a period of delay time T. By doing so, the node can withdraw its IB without the need for an additional on-chain transaction, thus saving transaction costs.

Case B: Node Rectify. In this situation, when the Center submits expired fund information, the node has the option to appeal to MPC+ by submitting the latest information along with the corresponding signature from the Center. MPC+ performs a series of checks on the submitted data. If all the checks pass successfully, the Center sets its *withdrawn* state to *True* and MPC+ proceeds to transfer funds based on the new message.

After the delay period T, if some nodes did not respond, the Center can proceed to submit the *TimeoutMsg* to MPC+ in order to officially withdraw the corresponding IBs. In **Procedure 4**, MPC+ performs a series of checks on the submitted information. If any discrepancies or errors are found during the checks, all operations will be rolled back to maintain the integrity of the system. It is possible that some nodes may have already agreed to the Center's application during the delay period. To prevent unnecessary rollback

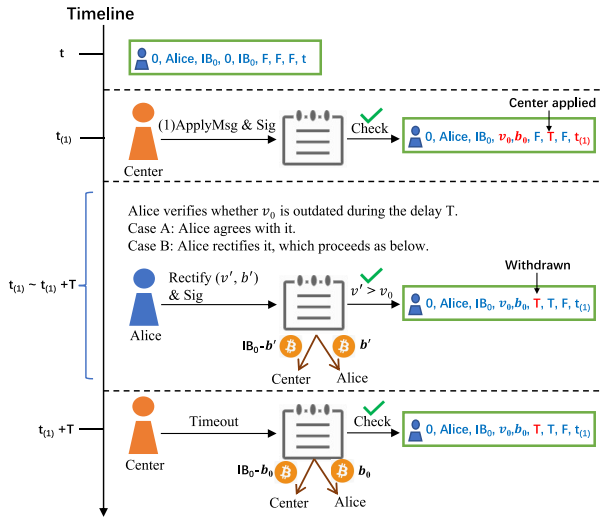


Fig. 6. An example of Apply Withdrawal.

operations, MPC+ checks whether the *withdrawn* state of the IBs is already set to *True*. This check ensures that if the withdrawal has already been processed, it will not be rolled back. If all the checks are correct, MPC+ transfers the IBs according to the applications and sets their *withdrawn* states to *True* to prevent any potential replay attacks.

The entire timeline of an Apply Withdrawal process is illustrated in Fig. 6. In some cases, Alice ($\mathcal{N}_0$) may not respond to the Center's *CoWithdrawalMsg*. In such situations, the Center has the option to apply for the withdrawal of Alice's fund, IB_0. However, there is a possibility that the Center may submit outdated information. Therefore, Alice needs to verify the information provided by the Center during the delay period. Alice can either agree with the information or rectify it by submitting evidence of a higher version. If Alice still does not respond after the delay period, the Center can proceed to officially withdraw IB_0.

c) Node apply withdrawal: If a node wishes to withdraw its own IB back to its account but encounters uncooperative behavior from the Center, it can apply for the withdrawal by submitting an *ApplyMsg* to MPC+. It is important to note that, apart from the Center, a node can only deal with its own IB. In a scenario where the funds initially belong to the node, but the Center is uncooperative in facilitating off-chain transactions, the node cannot submit the Center's signature corresponding to version 0. In such cases, MPC+ performs a special process when dealing with version 0. As outlined in **Procedure 5**, MPC+ conducts a series of checks on the submitted application. If all the checks pass successfully, MPC+ records the state according to the application.

During the delay period T, if one application cannot pass the verification, the Center sends the latest information to MPC+ for further processing. The process is similar to the rectify process described in **Procedure 3** (Case B). After the delay period T and if the Center does not invoke a rectify process, either the Center or the node can send a *TimeoutMsg* to MPC+ to officially withdraw the node's funds. Similarly, the *withdrawn* state of this fund will be set to *True* before transferring to prevent any potential replay attacks.

Procedure 5: Node Apply Withdrawal

Input: *ApplyMsg* $(index_i, v_i, b_i)$ and $Sig_{\mathcal{C}}$
if *sender is not* $\mathcal{N}_i$, *or* $\mathcal{N}_i.\mathcal{C}Apply$ = *True,or* $\mathcal{N}_i.\mathcal{N}Apply$ = *True* **then**
  return;
if v_i = 0 **then**
  $\mathcal{N}_i.\mathcal{N}Apply \leftarrow True$;
  $\mathcal{N}_i.timestamp \leftarrow now$;
elseif $b_i \notin [0, IB_i]$, *or* $Sig_{\mathcal{C}}$ *is unmatched* **then**
  return;
else
  $\mathcal{N}_i.ver \leftarrow v_i$;
  $\mathcal{N}_i.bal \leftarrow b_i$;
  $\mathcal{N}_i.\mathcal{N}Apply \leftarrow True$;
  $\mathcal{N}_i.timestamp \leftarrow now$;
Output: *ApplyInfo*

V. Formal Description of MPC+

This section first describes the security model of MPC+. Then, we present the ideal functionality for MPC+ before analyzing its security. Lastly, the role of the Center is discussed.

A. The Security Model

We formalize the security properties of MPC+ using the universal composability (UC) framework [41]. In this framework, a protocol is considered *UC secure* if the execution of the protocol is indistinguishable between the ideal world and the real world. The MPC+ contract functionality $\mathcal{M}$ is formalized to maintain contract instances with a $\mathcal{M}$-hybrid real world, where each MPC+ contract is identified by an identifier *id*. When receiving messages from parties, $\mathcal{M}$ accesses the ledger based on predefined conditions. The model of MPC+ is defined as $\mathcal{M}$ over a set of parties $\mathcal{P} = \{\mathcal{C}, \mathcal{N}_0, \mathcal{N}_1, \ldots, \mathcal{N}_n\}$, where $\mathcal{C}$ is the creator of MPC+ and $\mathcal{N}_i$ is the i-th node joining $\mathcal{M}$. We denote $\mathcal{N}^P.state$ as the view of party P on $\mathcal{N}.state$. The transfer mechanics are modeled through the global ideal functionality ledger $\mathcal{L}$, and we denote the transfer event "P_1 transfers x coins to P_2 on-chain" as "$P_1 \overset{x}{\rightharpoonup} P_2$", where P can be a node or a contract.

In the security model, the adversary $\mathcal{A}$ may corrupt parties with full control over the party's actions and can get the transmitted messages but cannot intercept/change them. The transmitted messages come from the synchronous communication network where parties communicate in rounds. We denote the event that "send message m to P in round t" as "$m \overset{\mathrm{t}}{\hookrightarrow} P$" and "receive message m from P in round t" as "$m \overset{\mathrm{t}}{\hookleftarrow} P$". The transmission of one message between two parties requires 1 round, which represents the message sent in round t reaches the receiver in round $t+1$. For simplicity, we assume the computations/communications with the environment take no round. The security is formally defined as follows.

Definition 1: Regarding the ledger $\mathcal{L}$ and the security parameter $\lambda \in \mathbb{N}$, MPC+ is a protocol running in the $\mathcal{M}$-hybrid world. MPC+ realize the ideal functionality $\mathcal{F}_{\mathcal{M}}$

(A) Creation

Upon receiving $(\texttt{create}, \mathcal{M}) \overset{t}{\hookleftarrow} P$ for creating $\mathcal{M}$, $\mathcal{F}_{\mathcal{M}}$ creates a $\mathcal{M}$ with identifier id, and records P as the only $\mathcal{C}$ of $\mathcal{M}_{id}$ eternally. After that, $\mathcal{F}_{\mathcal{M}}$ adds $\mathcal{M}_{id}$ to $\sum$, and sends $(\texttt{created}, \mathcal{M}, id) \overset{t+\Delta}{\hookrightarrow} \mathcal{C}$.

(B) Joining

Upon receiving $(\texttt{join}, \mathcal{M}, id, x) \overset{t}{\hookleftarrow} P$ for joining $\mathcal{M}_{id}$, $P \overset{x}{\rightharpoonup} \mathcal{M}_{id}$. $\mathcal{F}_{\mathcal{M}}$ assigns a new index i to represent P, and adds $\mathcal{N}_i = \{addr : P, IB : x, state : (0, x, False, False, False, now)\}$ to Nodes. Finally, $\mathcal{F}_{\mathcal{M}}$ sends $(\texttt{joined}, \mathcal{M}, id, i) \overset{t+\Delta}{\hookrightarrow} P$ and $\mathcal{C}$.

(C) Update

Assume the initiator of IB's update is P and the responder is Q.

Upon receiving $(\texttt{update}, id, i, \theta) \overset{t}{\hookleftarrow} P$, where θ represents the transferred amount, $(\texttt{update-req}, id, i, \theta) \overset{t+1}{\hookrightarrow} Q$.

1) If $(\texttt{update-ok}) \overset{t+2}{\hookleftarrow} Q$, then $IB_i.P := IB_i.P - \theta$ and $IB_i.Q := IB_i.Q + \theta$, and the update is completed.
2) Otherwise, set θ as IB_i's pending update, and stop.

(D) Bind Co-Withdrawal

Upon receiving $(\texttt{bind-cowithdrawal}, MsgS, SigS) \overset{t}{\hookleftarrow} \mathcal{C}$, $\mathcal{F}_{\mathcal{M}}$ splits $MsgS$ and $SigS$ into seperate $msg_i := (id, i, val)$ and corresponding sig_i orderly, where i directs the specific fund IB_i and node $\mathcal{N}_i$ in $\mathcal{M}_{id}$, and val is the fund that belongs to $\mathcal{N}_i$. $\mathcal{F}_{\mathcal{M}}$ checks every msg_i and sig_i orderly as follow.

1) If $val \notin [0, IB_i]$, or $\mathcal{N}_i.withdrawn = True$, or sig_i is unmatched, $\mathcal{F}_{\mathcal{M}}$ rolls back and terminates procedure.
2) $\mathcal{F}_{\mathcal{M}}$ sets $\mathcal{N}_i.withdrawn = True$, $\mathcal{M}_{id} \overset{val}{\rightharpoonup} \mathcal{N}_i$, $\mathcal{M}_{id} \overset{IB_i - val}{\rightharpoonup} \mathcal{C}$.

(E) Bind Apply Withdrawal, and Respond

Upon receiving $(\texttt{bind-apply-withdrawal}, MsgS, SigS) \overset{t}{\hookleftarrow} \mathcal{C}$, $\mathcal{F}_{\mathcal{M}}$ performs the following operations.

1) $\mathcal{F}_{\mathcal{M}}$ splits $MsgS$ and $SigS$ into seperate $msg_i := (id, i, ver, val)$ and corresponding sig_i orderly, where i directs the specific node $\mathcal{N}_i$ in $\mathcal{M}_{id}$. $\mathcal{F}_{\mathcal{M}}$ then proceeds to check each msg_i and sig_i orderly, following the procedure outlined below.
   a) If $\mathcal{N}_i.\mathcal{C}Apply = True$, or $val \notin [0, IB_i]$, or sig_i is unmatched, $\mathcal{F}_{\mathcal{M}}$ rolls back and terminate this procedure.
   b) If $\mathcal{N}_i.\mathcal{N}Apply = False$, $\mathcal{F}_{\mathcal{M}}$ sets $\mathcal{N}_i.\mathcal{C}Apply = True$, $\mathcal{N}_i.ver = ver$, $\mathcal{N}_i.bal = val$.
2) If $(\texttt{center-apply-ok}) \overset{t+\Delta}{\hookleftarrow} \mathcal{Z}$, the related nodes perform verification based on the following scenarios.
   a) Upon receiving $(\texttt{node-agree}, i) \overset{t_1 > t+\Delta}{\hookleftarrow} \mathcal{N}_i$, $\mathcal{N}_i \overset{\delta}{\rightharpoonup} \mathcal{L}$. $\mathcal{F}_{\mathcal{M}}$ terminates this procedure when $\mathcal{N}_i.\mathcal{C}Apply = False$, or $\mathcal{N}_i.withdrawn = True$. Otherwise, $\mathcal{F}_{\mathcal{M}}$ sets $\mathcal{N}_i.withdrawn = True$, $\mathcal{M}_{id} \overset{\mathcal{N}_i.bal}{\rightharpoonup} \mathcal{N}_i$, $\mathcal{M}_{id} \overset{IB_i - \mathcal{N}_i.bal}{\rightharpoonup} \mathcal{C}$.
   b) Upon receiving $(\texttt{node-rectify}, id, i, ver', val', sig_{\mathcal{C}}) \overset{t_2 > t+\Delta}{\hookleftarrow} \mathcal{N}_i$, $\mathcal{N}_i \overset{\delta}{\rightharpoonup} \mathcal{L}$. $\mathcal{F}_{\mathcal{M}}$ terminates this procedure if $\mathcal{N}_i.\mathcal{C}Apply = False$, or $\mathcal{N}_i.withdrawn = True$, or $ver' \leq \mathcal{N}_i.ver$, or $val' \notin [0, IB_i]$, or $sig_{\mathcal{C}}$ is unmatched. Otherwise, $\mathcal{F}_{\mathcal{M}}$ sets $\mathcal{N}_i.withdrawn = True$, $\mathcal{M}_{id} \overset{val'}{\rightharpoonup} \mathcal{N}_i$, $\mathcal{M}_{id} \overset{IB_i - val'}{\rightharpoonup} \mathcal{C}$.
   c) Upon receiving $(\texttt{bind-timeout-withdrawal}, MsgS) \overset{t_3 > t+\Delta+T}{\hookleftarrow} \mathcal{C}, \mathcal{C} \overset{\delta}{\rightharpoonup} \mathcal{L}$. $\mathcal{F}_{\mathcal{M}}$ splits $MsgS$ into seperate $msg_i := (id, i)$, and checks every msg_i orderly as follows.
      i) If $\mathcal{N}_i.\mathcal{C}Apply = False$, $\mathcal{F}_{\mathcal{M}}$ rolls back and terminate this procedure.
      ii) If $\mathcal{N}_i.withdrawn = False$, $\mathcal{F}_{\mathcal{M}}$ sets $\mathcal{N}_i.withdrawn = True$, $\mathcal{M}_{id} \overset{\mathcal{N}_i.bal}{\rightharpoonup} \mathcal{N}_i$, $\mathcal{M}_{id} \overset{IB_i - \mathcal{N}_i.bal}{\rightharpoonup} \mathcal{C}$.

Fig. 7. Ideal functionality $\mathcal{F}_{\mathcal{M}}$.

if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$:

$$\text{EXEC}^{\mathcal{L},\mathcal{M}}_{\pi,\mathcal{Z},\mathcal{A}}(\lambda) \approx \text{IDEAL}^{\mathcal{L}}_{\mathcal{F}_{\mathcal{M}},\mathcal{Z},\mathcal{S}}(\lambda)$$

where $\text{EXEC}^{\mathcal{L},\mathcal{M}}_{\pi,\mathcal{Z},\mathcal{A}}(\lambda)$ is the output of the environment $\mathcal{Z}$ when interacting with the adversary $\mathcal{A}$ and the protocol π in the $\mathcal{M}$-hybrid world, $\text{IDEAL}^{\mathcal{L}}_{\mathcal{F}_{\mathcal{M}},\mathcal{Z},\mathcal{S}}(\lambda)$ denotes the output when interacting with the ideal functionality $\mathcal{F}_{\mathcal{M}}$ and the simulator $\mathcal{S}$, $\approx$ denotes computational indistinguishability.

B. Ideal Functionality $\mathcal{F}_{\mathcal{M}}$ for MPC+

The ideal functionality $\mathcal{F}_{\mathcal{M}}$ interacts with the global ledger $\mathcal{L}$, the parties in the set $\mathcal{P}$, and the simulator $\mathcal{S}$. All created MPC+ instances are listed in the set $\sum$. The ideal functionality $\mathcal{F}_{\mathcal{M}}$ is illustrated in Fig. 7, depicting five main procedures: (A) **Creation** handles the opening of an MPC+ instance when triggered by $\mathcal{C}$. (B) **Joining** stores the information of the newly joined node. (C) **Update** performs the redistribution of funds, specifically the off-chain transactions. (D) **Bind Withdrawal** withdraws funds according to the messages submitted by $\mathcal{C}$. (E) **Bind Apply Withdrawal, and Respond** processes the

Assume the following messages concerning the MPC+ $\mathcal{M}$ with an identifier id.

(A) Creation

If $\mathcal{C}$ is honest, upon $\mathcal{C}$ sending $(\texttt{create}, \mathcal{M}) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$, send $(\texttt{create}, \mathcal{M}) \overset{t}{\hookrightarrow} \mathcal{M}(id)$ on behalf of $\mathcal{C}$. Otherwise, upon $\mathcal{C}$ sending $(\texttt{create}, \mathcal{M}) \overset{t}{\hookrightarrow} \mathcal{M}(id)$, send $(\texttt{create}, \mathcal{M}) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$ on behalf of $\mathcal{C}$.

(B) Joining

If P is honest, upon P sending $(\texttt{join}, \mathcal{M}, id, x) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$, send $(\texttt{join}, \mathcal{M}$, id, x, i$) \overset{t}{\hookrightarrow} \mathcal{M}(id)$ on behalf of P, and $\mathcal{M}(id)$ assigns a new index i to represent P. Otherwise, upon P sending $(\texttt{join}, \mathcal{M}$, id, x, i$) \overset{t}{\hookrightarrow} \mathcal{M}(id)$, send $(\texttt{join}, \mathcal{M}, id, x) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$ on behalf of P.

(C) Update

Assume the initiator of IB_i's update is P and the responder is Q.

Case: P is honest and Q is corrupt

Upon P sending $(\texttt{update}, id, i, \theta) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$, sign the $msg := (id, i, \mathcal{M}^P.ver + 1, \mathcal{M}^P.bal + \theta)$, and send $(\texttt{update}, msg, \sigma_P) \overset{t}{\hookrightarrow} Q$ on behalf of P. If Q signs on msg, and sends σ_Q to P in round $t+1$, send $(\texttt{update-ok}) \overset{t+1}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$ on behalf of Q. Otherwise, pend this IB update on behalf of P.

Case: P is corrupt and Q is honest

Upon P sending $(\texttt{update}, msg = (id, i, ver, bal), \sigma_P) \overset{t}{\hookrightarrow} Q$. If $ver = \mathcal{M}^Q.ver + 1$ and $bal = \mathcal{M}^Q.bal + \theta$, send $(\texttt{update}, id, i, \theta) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$ on behalf of P. Otherwise, stop. If Q sends $(\texttt{update-ok}) \overset{t+1}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$, generate Q's signature σ_Q on msg and send it to P on behalf of Q.

(D) Bind Co-Withdrawal

Case: $\mathcal{C}$ is honest

Upon $\mathcal{C}$ sending $(\texttt{bind-cowithdrawal}, MsgS, SigS) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$, send $(\texttt{bind-cowithdrawal}, MsgS, SigS) \overset{t}{\hookrightarrow} \mathcal{M}$ on behalf of $\mathcal{C}$. Process each $Msg = (id, i, b_i)$ and Sig_i orderly separately depending on the following situations.

1) If $\mathcal{N}_i.withdrawn = True$ or $b_i \notin [0, IB_i]$ or Sig_i is unmatched, send $(\texttt{roll-back-and-return}) \overset{t+\Delta}{\longrightarrow} \mathcal{C}$ on behalf of $\mathcal{M}$.
2) If all information in $MsgS$ and $SigS$ are checked correct, send $(\texttt{bind-cowithdrawal-ok}) \overset{t+\Delta}{\longrightarrow} \mathcal{C}$ on behalf of $\mathcal{M}$.

Case: $\mathcal{C}$ is corrupt

Upon $\mathcal{C}$ sending $(\texttt{bind-cowithdrawal}, MsgS, SigS) \overset{t}{\hookrightarrow} \mathcal{M}$, send $(\texttt{bind-cowithdrawal}, MsgS, SigS) \overset{t}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$ on behalf of $\mathcal{C}$. If all the $MsgS$ and $SigS$ are checked correct, send $(\texttt{bind-cowithdrawal-ok}) \overset{t+\Delta}{\longrightarrow} \mathcal{C}$ on behalf of $\mathcal{M}$. Otherwise, send $(\texttt{roll-back-and-return}) \overset{t+\Delta}{\longrightarrow} \mathcal{C}$ on behalf of $\mathcal{M}$.

Fig. 8. Simulation: (A) Creation, (B) Joining, (C) Update and (D) Bind Co-Withdrawal.

applications by $\mathcal{C}$ and handles the responses triggered by the corresponding nodes or by $\mathcal{C}$ after a period of delay T.

We focus on two expected security goals as outlined below.

Consensus on off-chain update: Every IB is independent, and the update of IB_i can only be executed by $\mathcal{C}$ and $\mathcal{N}_i$. When both users are honest, it takes two rounds to confirm the result.

Consensus on on-chain withdrawal: The IB can be withdrawn back to the account through two main methods. 1) *Bind Withdrawal*: This procedure can only be triggered by $\mathcal{C}$, and it is guaranteed to be completed within 1Δ time. 2) *Apply Withdrawal*: Any user can apply to withdraw their relevant IB at any time, and the request lasts for 1Δ time. The corresponding responder can either agree to it or rectify it within a total of 2Δ time. Alternatively, they can withdraw the funds after a delay of T within a total of $T + 2\Delta$ time.

C. Security Analysis of MPC+

Theorem 1: The MPC+ protocol *UC-securely* realizes the ideal functionality $\mathcal{F}_{\mathcal{M}}$ in the $\mathcal{M}$-hybrid world.

Proof: To ensure the indistinguishability between the real world and the ideal world, the simulator $\mathcal{S}$ interacts with the ideal functionality $\mathcal{F}_{\mathcal{M}}$ on behalf of the corrupted parties, and interacts with the corrupted parties on behalf of the honest participants and the contract functionality $\mathcal{M}$. Our main security goal is to protect the funds of the honest parties, whether they are on-chain or off-chain. As shown in Fig. 8 and Fig. 9, we illustrate how the simulator $\mathcal{S}$ handles different procedures.

Creation and Joining: Simulations on the two procedures are straightforward. Firstly, $\mathcal{S}$ interacts with $\mathcal{M}$ on behalf of honest parties, following their instructions sent to $\mathcal{F}_{\mathcal{M}}$. Then, $\mathcal{S}$ interacts with F on behalf of the corrupted participants, following their instructions sent to $\mathcal{M}$.

Update: The update process involves two parties and is executed off-chain, with each message delivery taking one round. First, $\mathcal{S}$ generates the signature of the initiator based on their payment and sends it to the responder on behalf of the initiator. These nodes perform the transaction based on their respective views. If any information is verified incorrect, the

(E) Apply Withdrawal, and Respond

Assume the applicant is P and the responder is Q, where P is $\mathcal{C}$ or $\mathcal{N}_i$, while Q is the other one.

Case: P is honest and Q is corrupt

Upon P sending $(\texttt{apply-withdraw}, ApplyMsg, Sig) \overset{\text{t}}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$, $(\texttt{apply-withdraw}, IB_i^P) \overset{\text{t}}{\hookrightarrow} \mathcal{M}$ on behalf of P. Proceed as follows,

1) If Q sends $(\texttt{agree/rectify}) \xhookrightarrow{\tau_1 > t+\Delta} \mathcal{M}$, send $(\texttt{agree/rectify}) \xhookrightarrow{\tau_1 > t+\Delta} \mathcal{F}_{\mathcal{M}}$ on behalf of Q. Distinguish the following cases,
   a) If IB_i is withdrawn, send $(\texttt{return}) \xhookrightarrow{\tau_1+2\Delta} Q$ on behalf of $\mathcal{M}$.
   b) Otherwise, send $(\texttt{withdrawn}) \xhookrightarrow{\tau_1+2\Delta} P$ on behalf of $\mathcal{M}$.
2) If P sends $(\texttt{timeout}) \xhookrightarrow{\tau_2 > t+\Delta+T} \mathcal{F}_{\mathcal{M}}$, send $(\texttt{timeout}, i) \xhookrightarrow{\tau_2 > t+\Delta+T} \mathcal{M}$ on behalf of P. Refund IB_i, and send $(\texttt{withdrawn}) \xhookrightarrow{\tau_2+\Delta} P$ on behalf of $\mathcal{M}$.

Case: P is corrupt and Q is honest

When P sends $(\texttt{apply-withdraw}, IB_i^P) \overset{\text{t}}{\hookrightarrow} \mathcal{M}$, send $(\texttt{apply-withdraw}, ApplyMsg, Sig) \overset{\text{t}}{\hookrightarrow} \mathcal{F}_{\mathcal{M}}$ on behalf of P. Otherwise, stop. If Q sends $(\texttt{agree/rectify}) \xhookrightarrow{\tau_1 > t+\Delta} \mathcal{F}_{\mathcal{M}}$, send $(\texttt{agree/rectify}) \xhookrightarrow{\tau_1 > t+\Delta} \mathcal{M}$ on behalf of Q. Distinguish the following situations,

1) If IB_i is withdrawn, send $(\texttt{return}) \xhookrightarrow{\tau_1+2\Delta} Q$ on behalf of $\mathcal{M}$.
2) Otherwise, send $(\texttt{withdrawn}) \xhookrightarrow{\tau_1+2\Delta} P$ on behalf of $\mathcal{M}$.

Fig. 9. Simulation: (E) Apply Withdrawal and Respond.

update is put on hold. Otherwise, $\mathcal{S}$ generates the signature of the responder and sends it to the initiator on behalf of the responder. As a result, $\mathcal{F}_{\mathcal{M}}$ maintains the same balance distribution as the two parties.

Bind Co-Withdrawal: In this procedure, $\mathcal{S}$ simulates the $\mathcal{C}$ to withdraw multiple funds with the *CoWithdrawalMsgS* and *SigS*. Since $\mathcal{M}$ will roll back all operations if something is checked wrong, $\mathcal{S}$ cannot arbitrarily generate the *states* of parties. Instead, $\mathcal{S}$ can instruct $\mathcal{M}$ to perform a series of checks on these messages. If all information is correct, funds can be transferred accordingly. However, if any information is found to be incorrect, all operations will be rolled back, and $\mathcal{S}$ sends the *fail* information to $\mathcal{C}$ on behalf of $\mathcal{M}$.

Apply Withdrawal, and Respond. In the bind operation, which handles multiple funds independently, let's focus on the application of one fund for simplicity. $\mathcal{S}$ first sends the *ApplyMsg* to $\mathcal{F}_{\mathcal{M}}$ on behalf of *P*, and then handles different situations that may occur. The *ApplyMsg* can be either the latest or expired. Since $\mathcal{F}_{\mathcal{M}}$ holds the same fund distribution for all parties, $\mathcal{S}$ can instruct the responder to either agree on the distribution or rectify it if necessary. The responder has the option to choose not to respond. In this case, $\mathcal{S}$ sends the *TimeoutMsg* to $\mathcal{M}$ on behalf of *P* after a delay of *T* to officially withdraw the funds.

With the simulations described above, the MPC+ protocol achieves computational indistinguishability between the ideal world and the real world. ■

D. Discussion

For the presence of a Center in MPC+, it is important to note that in routine payment scenarios, such as Cafe or similar establishments, there is often a need for a central entity that frequently receives funds from multiple nodes. In other words, the perceived centralization of MPC+ is a result of the Center node's role in regularly receiving funds from multiple nodes. In practice, MPC+ does not contradict the principles of decentralization, as the process of locking funds into MPC+ or returning locked funds back to individual accounts is determined by the entire blockchain network, without relying on any central trusted third-party. Furthermore, each initial balance in MPC+ operates independently and is only accessible to the Center and the node that locked it. MPC+ does not rely on trust between the Center and nodes, and the off-chain transactions within MPC+ do not require validation from a central trusted third-party.

Additionally, the MPC+ system can be corrupted if the Center is offline during transactions or experiences a DDoS attack. However, it is important to note that such a corrupted situation is not unique to MPC+ and can occur in other solutions [25], including the main net of a blockchain. Furthermore, in MPC+, even if the Center misbehaves, the worst-case scenario is that nodes are unable to make further off-chain transactions with no financial loss. In such a situation, each node can choose to apply for the withdrawal of its funds, ensuring the security of its funds.

VI. Implementation and Evaluation

The main goal in designing MPC+ is to minimize the number of on-chain transactions while maintaining the properties of instant transactions and compatibility. In this section, we implement the MPC+ smart contract on Ethereum using the Turing-complete language Solidity to determine the execution cost of the Center and every node under different scenarios. The evaluation criteria for fees associated with on-chain transactions on Ethereum is referred to as gas. The amount of gas consumed heavily depends on the volume of information

TABLE III
EXECUTION COST COMPARISON

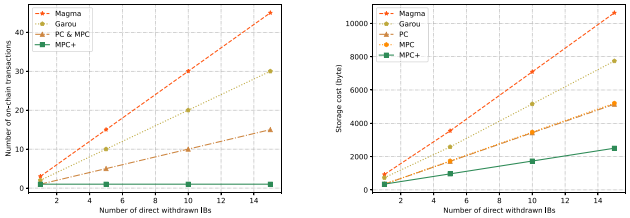
Behavior	On-chain tx	Gas /10^3	Gas payer
create n PCs	n	$1579.1n$	n nodes
create MPC	1	2970.8	Center
n nodes join MPC	n	$78.6n$	n nodes
create MPC+	1	3052.4	Center
n nodes join MPC+	n	$78.6n$	n nodes
create Magma	1	3035.0	Center
n nodes join Magma	n	$41.5n$	n nodes
tx in Garou:			
(optimistic)	0	0	—
(pessimistic)	$1+m$	$10.9n$	$1+m$ nodes[1]
tx in Magma:			
(optimistic)	0	0	—
(pessimistic)	1	$179.5n+93.8$	1 node[2]
direct tx in PC	0	0	—
indirect tx in PCN:			
(optimistic)	0	0	—
(pessimistic)	2	149.6	payer&payee[3]
direct tx in MPC+	0	0	—
indirect tx in MPC+:			
(optimistic)	0	0	—
(pessimistic)	2	140.3	payer&payee[3]

[1] 1 honest node challenges, and m malicious nodes respond.
[2] 1 node reports the Center's misbehave.
[3] The payee challenges, and the payer responds.

and the complexity of computations involved. Additionally, each on-chain transaction records essential information and incurs a basic fee of 21,000 gas, which is paid to compensate the miner for verifying the accuracy of the transaction [38]. Having more on-chain transactions results in increased basic overhead.

The execution costs, excluding withdrawal, under different schemes are presented in Table III. In the first phase, the advantage of MPC+ is not particularly evident, as the gas cost of creating MPC+ is 3052482, which is only lower than that of creating n PCs. This is because MPC+ enhances the withdrawal logic based on the previous version of MPC [37]. Although the costs, excluding withdrawal, for MPC+ exceed those of the previous solutions, the Center and nodes will reap greater benefits in the subsequent phases.

In the update (off-chain transaction) phase, if all parties are honest (optimistic case), the updates in all solutions can be completed off-chain immediately without incurring any gas fees. Otherwise (pessimistic case), on-chain challenges may be required for the update process. In the case of MPC+, since there is only one IB and the direct update involves only two parties, the off-chain transaction can be confirmed immediately even if one party misbehaves. On the other hand, in Garou [24] or Magma [25], all participants' confirmations are necessary. If any party fails to confirm, the honest parties would need to issue an on-chain challenge to complete the update, which could result in a large number of on-chain transactions or significant fees. The pessimistic cases of indirect updates in PCN and MPC+ occur when the payer denies paying the funds that were promised to the payee. In such scenarios, the challenge cost in MPC+ is comparable to that of PC and PCN, but it outperforms Garou and Magma in terms of challenge costs.



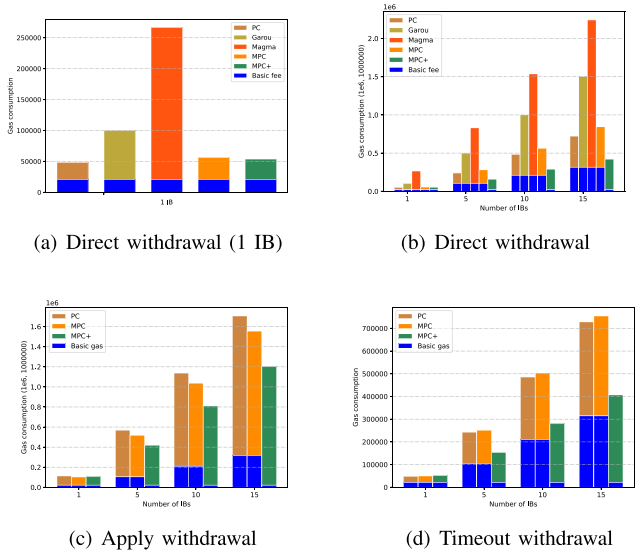
(a) Number of on-chain transactions (b) Storage cost

Fig. 10. The effect of the number of IBs on direct withdrawal.

Indeed, the withdrawal logic in these solutions differs from one another. For instance, in Garou and Magma, a single fund can be transferred to any participant since all participants share the total funds through updates. On the other hand, in PC, MPC, and MPC+, each fund is specifically owned by the Center and the corresponding participant. Considering the most efficient way, the direct withdrawal represents a scenario where all parties cooperate to withdraw the funds. In Garou and Magma, the direct withdrawal of a single fund is initiated by one party's application. It requires confirmations from others, similar to the apply withdrawal and timeout withdrawal in PC, MPC, and MPC+. In the latter three solutions, the direct withdrawal of a single fund can be completed in a single on-chain transaction.

Fig. 10 illustrates the changes in the number of on-chain transactions and storage costs for each scheme as the number of direct withdrawal IBs increases. In the case of MPC+, which processes multiple IBs simultaneously, the number of on-chain transactions remains fixed at 1, regardless of the number of IBs. Conversely, in other solutions, the number of on-chain transactions increases linearly with the addition of IBs. As depicted in Fig. 10(b), the increase in the number of on-chain transactions leads to a corresponding increase in storage costs for basic information. MPC+ outperforms other solutions regarding storage costs due to its efficient update logic and fewer on-chain transactions. Ethereum consumes an average of 178KB of storage overhead per block [42], primarily attributed to the data of packaged on-chain transactions. However, the update logic in Garou and Magma requires the signatures of all participants, and the withdrawal algorithms in these solutions also require all signatures. Each signature requires 130 bytes (hexadecimal). Therefore, the blockchain scalability of MPC+ is improved by reducing the number of on-chain transactions and lowering storage costs.

Respecting the fees for the withdrawal operations. We first test the fees for direct withdrawal by examining the fees for withdrawing 1 IB, as shown in Fig. 11(a). Since MPC and MPC+ adopt a similar update logic to PC, the withdrawal costs for 1 IB in these three solutions are close. However, in Garou and Magma, where the update logic requires consensus from all participants, the gas fees increase significantly. Additionally, each on-chain transaction incurs a basic fee of 21000 gas, represented by the blue portion below each column. Next, we test the fees for direct withdrawal of 5, 10, and 15 IBs, respectively, and the comparison of execution costs is shown in Fig. 11(b). The gas consumption in MPC+



(a) Direct withdrawal (1 IB)

(b) Direct withdrawal

(c) Apply withdrawal

(d) Timeout withdrawal

Fig. 11. The gas consumption comparison under different solutions is depicted above, where the blue part below each column represents the basic gas fee for on-chain transactions. The direct withdrawal means that all participants cooperate to withdraw funds. In MPC+, the basic gas fee is only paid once, regardless of the number of IBs, while the other solutions pay the basic gas fee the same number of times as the number of IBs.

is much lower than that in PC and MPC because MPC+ only pays the basic gas fee once, regardless of the number of IBs. In contrast, the other solutions pay the 21000 gas basic fee for each IB. Since Garou and Magma do not have functions such as apply withdrawal and timeout withdrawal, as seen in PC, MPC, and MPC+, we primarily illustrate the fees for applying withdrawal and timeout withdrawal in the latter three solutions in Fig. 11(c) and Fig. 11(d), respectively. In the case of MPC+, it only pays the basic fee once, which outperforms the other two solutions by reducing the number of on-chain transactions from $O(n)$ to $O(1)$.

VII. Conclusion and Future Work

This paper introduces MPC+, a smart contract that enables multiple nodes to engage in instant off-chain transactions. MPC+ is designed to cater to scenarios where one node frequently receives money from multiple nodes, such as Cafe, shops, and more. MPC+ seamlessly integrates with existing payment channel networks (PC&PCN), ensuring compatibility and interoperability. Furthermore, the main advantage of MPC+ over existing solutions is its ability to handle multiple funds simultaneously, effectively consolidating multiple on-chain transactions into a single transaction. We rigorously prove the security of MPC+ and implement it in Ethereum using the Solidity language. The experimental results demonstrate that MPC+ outperforms existing designs in terms of efficiency and effectiveness.

Although the likelihood of misconduct by the center is low, and the refund method is also included in MPC+, the fact remains that each off-chain transaction requires confirmation from a unique center. Therefore, if the center engages in misconduct, the MPC+ system will be compromised. To ensure the stability and security of the system, it is necessary to establish a secure decentralized node authorization protocol. In our future work, we plan to implement distributed multi-node payment channels using node delegation protocols and dynamic methods.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Bus. Rev.*, 2008, to be published.
[2] "Ethereum." 2024. [Online]. Available: https://www.ethereum.org/
[3] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Secur. Privacy, (SP)*, San Francisco, CA, USA, 2018, pp. 583–598.
[4] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-NG: A scalable blockchain protocol," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement., (NSDI)*, Santa Clara, CA, USA, 2016, pp. 45–59.
[5] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th ACM Symp. Oper. Syst. Princ., (SOSP)*, Shanghai, China, 2017, pp. 51–68.
[6] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, "A scalable multi-layer PBFT consensus for blockchain," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1146–1160, May 2021.
[7] X. Qu, S. Wang, Q. Hu, and X. Cheng, "Proof of federated learning: A novel energy-recycling consensus algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 2074–2085, Aug. 2021.
[8] M. Trillo, "Stress test prepares VisaNet for the most wonderful time of the year (2013)," Visa, Foster City, CA, USA, 2013.
[9] C. Decker and R. Wattenhofer, "A fast and scalable payment network with Bitcoin duplex micropayment channels," in *Proc. Symp. Self Stabiliz. Syst.*, Edmonton, AB, Canada, 2015, pp. 3–18.
[10] J. Poon and T. Dryja, "The Bitcoin lightning network: Scalable off-chain instant payments." 2016. [Online]. Available: https://www.bitcoinlightning.com/wpcontent/uploads/2018/03/lightning-network-paper.pdf
[11] "Raiden network." Accessed: Sep. 15, 2023. [Online]. Available: https://raiden.network/
[12] P. Li, T. Miyazaki, and W. Zhou, "Secure balance planning of off-blockchain payment channel networks," in *Proc. IEEE Int. Conf. Comput. Commun.*, Toronto, ON, Canada, 2020, pp. 1728–1737.
[13] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2017, pp. 439–453.
[14] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *Proc. IEEE Symp. Security Privacy*, San Francisco, CA, USA, 2019, pp. 106–123.
[15] V. Sivaraman et al., "High throughput cryptocurrency routing in payment channel networks," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implement.*, Santa Clara, CA, USA, 2020, pp. 777–796.
[16] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Proc. Int. Conf. Financ. Cryptogr. Data Secur.*, Cham, Switzerland, 2019, pp. 508–526.
[17] Y. Zhang and D. Yang, "Robustpay+: Robust payment routing with approximation guarantee in blockchain-based payment channel networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 4, pp. 1676–1686, Aug. 2021.
[18] S. Mercan, E. Erdin, and K. Akkaya, "Improving transaction success rate in cryptocurrency payment channel networks," *Comput. Commun.*, vol. 166, pp. 196–207, Jan. 2021.
[19] Y. Chen, Y. Ran, J. Zhou, J. Zhang, and X. Gong, "MPCN-RP: A routing protocol for blockchain-based multi-charge payment channel networks," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1229–1242, Jun. 2022.
[20] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, "CoinExpress: A fast payment routing mechanism in blockchain-based payment channel networks," in *Proc. Int. Conf. Comput. Commun. Netw.*, Zhejiang, China, 2018, pp. 1–9.
[21] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, Darmstadt, Germany, 2019, pp. 625–656.

[22] X. Jia, Z. Yu, J. Shao, R. Lu, G. Wei, and Z. Liu, "Cross-chain virtual payment channels," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 3401–3403, 2023.
[23] Y. Chen, X. Li, J. Zhang, and H. Bi, "Multi-party payment channel network based on smart contract," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 4, pp. 4847–4857, Dec. 2022.
[24] Y. Ye, Z. Ren, X. Luo, J. Zhang, and W. Wu, "Garou: An efficient and secure off-blockchain multi-party payment hub," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4450–4461, Dec. 2021.
[25] Z. Ge, Y. Zhang, Y. Long, and D. Gu, "Magma: Robust and flexible multi-party payment channel," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 6, pp. 5024–5042, Dec. 2023.
[26] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proc. ACM Conf. Comput. Commun. Secur.*, Toronto, ON, Canada, 2018, pp. 967–984.
[27] N. Narula, W. Vasquez, and M. Virza, "zkLedger: Privacy-preserving auditing for distributed ledgers," in *Proc. 15th USENIX Symp. Netw. Syst. Des. Implement.*, 2018, pp. 65–80.
[28] "Ripple." 2019. [Online]. Available: https://www.ripple.com/
[29] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, "DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement.*, Renton, WA, USA, 2022, pp. 493–512.
[30] E. B. Sasson et al., "Zerocash: Decentralized anonymous payments from Bitcoin," in *Proc. IEEE Symp. Security Privacy*, San Jose, CA, USA, 2014, pp. 459–474.
[31] J. Zhang, Y. Ye, W. Wu, and X. Luo, "Boros: Secure and efficient off-blockchain transactions via payment channel hub," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 1, pp. 407–421, Feb. 2023.
[32] H. Bi, Y. Chen, and X. Zhu, "A multipath routing for payment channel networks for Internet of Things microtransactions," *IEEE Internet Things J.*, vol. 9, no. 20, pp. 19670–19681, Oct. 2022.
[33] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing security and privacy in decentralized credit networks not every permissionless payment network requires a blockchain," in *Proc. 24th Annu. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2017, pp. 1–15.
[34] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *Proc. 25th Annu. Netw. Distrib. System Secur. Symp.*, San Diego, CA, USA, 2018, pp. 1–15.
[35] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *Proc. 26th Annu. Netw. Distrib. System Secur. Symp.*, San Diego, CA, USA, 2019, pp. 1–30.
[36] Q. Wang et al., "SorTEE: Service-oriented routing for payment channel networks with scalability and privacy protection," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 4, pp. 3764–3780, Dec. 2022.
[37] H. Lei, L. Huang, L. Wang, and J. Chen, "MPC: Multi-node payment channel for off-chain transactions," in *Proc. IEEE Int. Conf. Commun.*, Seoul, South Korea, 2022, pp. 4733–4738.
[38] G. Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project, Yellow Paper 151, 2014.
[39] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.
[40] M. I. Mehar et al., "Understanding a revolutionary and flawed grand experiment in blockchain: The DAO attack," *J. Cases Inf. Technol.*, vol. 21, no. 1, pp. 19–32, 2019.
[41] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. Annu. Symp. Found. Comput. Sci.*, Las Vegas, NV, USA, 2001, pp. 136–145.
[42] "Bitinfocharts." Accessed: Oct. 9, 2023.[Online]. Available: https://bitinfocharts.com/comparison/size-eth.html