Horcrux: Synthesize, Split, Shift and Stay Alive
Preventing Channel Depletion via Universal and Enhanced Multi-hop Payments

Anqi Tian[1,2,*], Peifang Ni[1,3,*], Yingzi Gao[1,2] and Jing Xu[1,3,†]
[1]Institute of Software, Chinese Academy of Sciences
[2]School of Computer Science and Technology, University of Chinese Academy of Sciences
[3]Zhongguancun Laboratory, Beijing, P.R.China
{anqi2021, peifang2020, yingzi2019, xujing}@iscas.ac.cn

***Abstract*—Payment Channel Networks (PCNs) have been highlighted as viable solutions to address the scalability issues in current permissionless blockchains. They facilitate off-chain transactions, significantly reducing the load on the blockchain. However, the extensive reuse of multi-hop routes in the same direction poses a risk of channel depletion, resulting in involved channels becoming unidirectional or even closing, thereby compromising the sustainability and scalability of PCNs. Even more concerning, existing rebalancing protocol solutions heavily rely on trust assumptions and scripting languages, resulting in compromised universality and reliability.**

In this paper, we present Horcrux, a universal and efficient multi-party virtual channel protocol without relying on extra trust assumptions, scripting languages, or the perpetual online requirement. Horcrux fundamentally addresses the channel depletion problem using a novel approach termed *flow neutrality*, which minimizes the impact on channel balance allocations during multi-hop payments (MHPs). Additionally, we formalize the security properties of Horcrux by modeling it within the Global Universal Composability framework and provide a formal security proof.

We implement Horcrux on a real Lightning Network dataset, comprising 10,529 nodes and 38,910 channels, and compare it to the state-of-the-art rebalancing schemes such as Shaduf [NDSS'22], Thora [CCS'22], and Revive [CCS'17]. The experimental results demonstrate that (1) the entire process of Horcrux costs less than 1 USD, significantly lower than Shaduf; (2) Horcrux achieves a 12%-30% increase in payment success ratio and reduces user deposits required for channels by 70%-91%; (3) the performance of Horcrux improves by $1.2x$-$1.5x$ under long-term operation; and (4) Horcrux maintains a nearly zero channel depletion rate, whereas both Revive and Shaduf result in thousands of depleted channels.

I. Introduction

Permissionless cryptocurrencies, such as Bitcoin [33], have been progressively emerging as an alternative payment method by leveraging blockchain technology. This technology relies on a decentralized consensus protocol to establish and maintain a secure, distributed ledger that tracks every transaction. Despite its numerous benefits, the scalability issue remains a significant hurdle limiting the widespread adoption and deployment of blockchain. For instance, in 2017, Bitcoin's daily transaction peak exceeded 420,000. However, Bitcoin can only process about 10 transactions per second due to limitations imposed by the permissionless nature of blockchain's consensus algorithm, whereas centralized solutions like Visa can handle over 50,000 transactions per second.

Payment Channel (PC) [1] has emerged as a promising solution to the scalability issue. Two users can cooperatively create a PC by locking coins in a jointly controlled address, allowing them to conduct several off-chain transactions by updating the channel state, which can be closed by publishing its latest state on-chain. While PCs effectively reduce the blockchain load, it is economically infeasible to establish such channels between all potential users, as it requires both parties to lock coins for each channel. To overcome this limitation, Payment Channel Networks (PCNs) have been proposed, connecting any two users via paths that utilize existing PCs as intermediaries. Implementations of PCNs are already widely used in practice, e.g., the Lightning Network (LN) [36], which is deployed upon Bitcoin, currently hosts over 150 million USD worth of bitcoins across more than 14,000 nodes and 64,000 channels.

Nevertheless, PCNs face the challenge of fund depletion in channels [35, 42]. The reuse of the same payment path leads to an accumulation of balances in channels along the path, preventing payments in that direction and potentially resulting in an imbalanced PCN structure. Moreover, when a channel depletes, all payments attempting to pass through it will fail, forcing users to choose more expensive routes. This results in higher transaction costs and limited throughput of entire PCN.

Current efforts focus on reviving depleted channels. A trivial approach of closing and then reopening a channel requires two costly on-chain transactions. Splicing [34] and LOOP [27] reduce this to one on-chain transaction, but they only support one-time refunding and require users to have sufficient on-chain funds. Pioneering works in rebalancing, represented by Revive [25], HIDE & SEEK [12], Thora [6] and Wiser [43], offer potential solutions by using coins from adjacent channels without affecting payments in the underlying channels. However, their reliance on cycle paths, extra trust assumptions, and minimal rebalancing amounts significantly undermine the practical availability of these off-chain rebalancing approaches. Non-cycle rebalancing, intro-

[*] Anqi Tian & Peifang Ni led efforts with equal contribution.
[†] Corresponding author.
This paper will appear in NDSS 2025.

duced by Shaduf [21], requires an on-chain setup that costs one transaction to bind two adjacent channels, allowing free coin transfers between them. Despite having minimal path restrictions and the ability to perform multiple rebalancing operations, Shaduf focuses on single-node rebalancing and requires smart contracts, limiting its application on scriptless blockchains like Bitcoin. In summary, let us overview the remaining hurdles and design challenges.

• *Inadequate fund depletion solving.* According to the real topology data of the LN [4], the average number of channels per user is 8.6, implying a highly connected network among users. Since a single multi-hop payment (MHP) can skew the fund allocations of all involved channels, it is critical to recognize that this balance skewness, introduced by payments, is the root of fund depletion. However, existing rebalancing protocols focus solely on addressing already depleted channels and have not made improvements to address this underlying issue, resulting in poor sustainability. This manifests in channels repeatedly facing depletion and closure risks, which severely limits the capacity and throughput of PCNs.

• *Reliance on smart contracts.* Smart contracts programmed over rich Turing-complete scripting languages guarantee the balance security of involved users and the coins shift between channels with a relatively simplified construction. However, it is not universally applicable, as the majority of existing blockchains do not support Turing-complete scripting languages (i.e., Bitcoin, Monero, Ripple, ZCash).

• *Weak reliability.* To ensure users' balance security, it is essential to atomically update the involved channels in the rebalancing process, requiring synchronization among the participating users in the protocol. For instance, in the case of Revive [25], a trusted third party acts as a fair leader to collect users' rebalancing demands and coordinate the rebalancing process. As an optimization, existing multi-channel update schemes employed in rebalancing protocols alternatively rely on the always-online assumption, utilizing users to supervise the blockchains and trigger disputes if necessary. However, it is challenging for users to remain persistently online due to various factors [11], i.e., hardware failures. Once a user fails to stay online, these protocols face security vulnerabilities, compromising their reliability and limiting their practical applicability.

The current state of affairs naturally leads to the following question: *Is it possible to design a rebalancing protocol that completely mitigates channel depletion without relying on extra trust assumptions for the involved users or the underlying blockchains (e.g., being always online or supporting scripting languages)?*

We affirmatively answer the above question in this work. We propose a targeted approach called *flow neutrality*, which enables users to securely transfer assets between adjacent channels while processing payments, completely eliminating the balance skewness introduced by MHPs. This fundamentally resolves the issue of channel depletion. Furthermore, by introducing *time-based fee incentive mechanism*, we achieve a secure Bitcoin-compatible multi-channel update protocol, while removing extra trust assumptions. Note that our protocol construction inherently resists domino attack [10], because we separate the unloading of Virtual Channels (VCs) from the closure of underlying channels. To further enhance reliability, we introduce the *Time Slicing* mechanism, allowing users to securely perform other payments or go offline, thereby eliminating the always-online assumption. Ultimately, we present Horcrux, a universal and efficient multi-party virtual channel protocol facilitating effective rebalancing without reliance on extra trust assumptions, scripting languages, or perpetual online connectivity. A comparison between Horcrux and state-of-the-art solutions is presented in Table I.

TABLE I: The comparisons among related works[1]

Protocol	Revive[3]	Thora[3]	Shaduf	**Our work**
On-chain setup[2]	✗	✗	✓	✓
Non-cycle construction	✗	✗	✓	✓
Bitcoin-compatible	✗	✓	✗	✓
Sustainability	✗	✗	✗	✓
Reliability	✗	✗	✗	✓

1 Revive [25], Thora [6], Shaduf [21].
[2] On-chain setup allows multiple rebalancing with one single on-chain transaction.
[3] Cycle construction provides one-time rebalancing but requires a leader and directed cyclic paths in PCNs.

Our contributions. In greater detail, the core contributions of Horcrux can be summarized as follows:

• *A comprehensive solution of fund depletion.* We analyze how MHPs contribute to the balance imbalance in channels along the routes and elucidate its severity, implying the imminent risk of channel depletion or even closure for involved users and the corresponding poor sustainability of PCNs. To address this issue, we introduce a novel approach *flow neutrality* that minimizes the impact on channel balance allocations during MHPs, fundamentally resolving the channel depletion issue and maintaining balanced and sustainable PCNs.

• *Universality enhancement.* Horcrux is designed for the UTXO model without relying on specific scripting features and only requires the bare minimum ability of digital signature verification support, ensuring compatibility with the majority of blockchains. Besides, this optimization of universality does not bring the complexity of construction. Whether the blockchain employs ECDSA, Schnorr, or BLS signature schemes, Horcrux can be implemented practically and efficiently.

• *Reliability enhancement.* Horcrux only necessitates the participation of intermediate users in the establishment and closure of virtual channels, without being involved in each payment. By utilizing the *Time Slicing* mechanism, intermediate users can securely go offline during designated time intervals, eliminating potential security vulnerabilities and significantly enhancing system reliability. Additionally, to mitigate the risk of domino attacks, Horcrux introduces a multi-channel update mechanism that separates the offloading process from the closing of channels, thereby preventing malicious users from closing channels prematurely.

• *Implementation and evaluation.* We provide a proof-of-concept implementation of Horcrux that respectively based on Schnorr/ECDSA signatures and simulation over a real LN dataset, encompassing $10,529$ nodes and $38,910$ channels [21], which demonstrates that Horcrux's on-chain costs several times lower than Shaduf [21], with the cost of entire protocol less than 1 USD. In terms of performance, we include measurements for network depletion as one of the evaluation criteria. We also test both the short-term and long-term performance of Horcrux by running 20 and 200 batches, respectively, with each batch containing 50,000 random transactions. Compared to existing works [25][21], Horcrux achieves a 12%-30% increase in payment success ratio, a reduction of 70%-91% in user deposits required for channels, and its performance improves by $1.2x$-$1.5x$ under long-term operation. More importantly, Horcrux maintains a nearly zero channel depletion rate throughout its operation, a noteworthy achievement that surpasses other protocols.

II. Preliminaries and background

In this section, we first introduce notations and preliminaries on UTXO-based blockchains. We then overview the basics of channel works including PCNs and virtual channels (VCs). We finally discuss the balance skewness in MHPs.

A. UTXO-based blockchains

We adopt the notation for UTXO-based blockchains(e.g., Bitcoin), which we shortly review next. In UTXO-based blockchains, the units of currency, i.e., the coins, exist in the outputs of transactions. We define such an output as a tuple $\theta := (cash, \phi)$. The component $\theta.cash$ contains the amount of coins stored in this output and $\theta.\phi$ defines the condition under which the coins can be spent. We say that a user U owns the coins in an output θ if $\theta.\phi$ contains only a signature verification w.r.t. the public key of U. For this, we use the notation pk_U.

In brief, a transaction in the UTXO model maps one or more existing outputs to a list of new outputs. The existing outputs are called transaction inputs. Formally, we use the public key pk to denote the input/output address and define a transaction tx as $tx[inputs, outputs, witness]$ to transfer the coins $\{v_1, \cdots, v_m\}$ from addresses $\{pk_1, \cdots, pk_m\}$ to $\{\widetilde{pk}_1, \cdots, \widetilde{pk}_\ell\}$, where $inputs := \{(pk_1, v_1), \cdots, (pk_m, v_m)\}$, $outputs := \{(\widetilde{pk}_1, \widetilde{v}_1), \cdots, (\widetilde{pk}_\ell, \widetilde{v}_\ell)\}$ and $witness := \{\sigma_1, \cdots, \sigma_m\}$ fulfills the spending condition of each input. Additionally, we focus on the scriptless blockchains, thus the witnesses σ_j $(j \in [1, m])$ is the signature w.r.t., the public keys pk_j.

Especially, we use a chart to visualize the transaction flow. As depicted in Fig. 1, we respectively use double-edge rectangles and single-edge rectangles to represent the transactions that are already confirmed on the blockchain and the ones that are not yet. Each transaction contains one or more boxes to denote its outputs and the corresponding amount of coins is written inside the output box. In addition, the spending condition of each output is written below the outputting arrow.

We use Fig. 1 to further illustrate the coin flow between the addresses controlled by a single user and addresses jointly controlled by multiple users, which is the basic transaction framework in the PC (network). Firstly, transactions tx_A with output coins α and tx_B with output coins β are respectively spent by users A and B who own secret keys sk_A and sk_B; while address $pk_{A,B}$ is controlled jointly by users A and B, then transaction $tx'_{A,B}$ is valid only if it has been multi-signed by users A and B; finally, the coins α' and β' return to the addresses respectively controlled by users A (i.e., address pk_A) and B (i.e., address pk_B), where $\alpha + \beta = \alpha' + \beta'$.

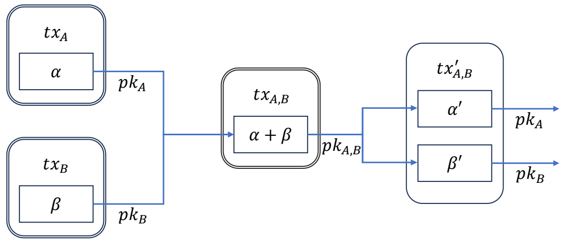


Fig. 1: The coins α and β flow respectively from addresses pk_A and pk_B to $pk_{A,B}$ (jointly controlled by A and B), then being spent by transaction $tx'_{A,B}$, where its two outputs with value α' and β' are respectively controlled by A and B.

B. Payment channel network

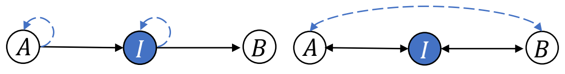
1) Payment channel: A PC enables two users to have arbitrarily many off-chain transactions when they have locked coins in a specific address jointly controlled by these two users, while only requiring two on-chain transactions respectively for opening and closing the channel during its lifespan. Since these on-chain transactions guarantee several off-chain transactions being executed securely, the PC technique efficiently improves the throughput of the underlying blockchain.

In this work, we use PCs in a black-box manner and refer the readers to work [8] for more details. On a high level (cf. Fig. 1), a PC $\gamma_{A,B}$ with capacity $\alpha + \beta$ between users A and B is successfully opened once the funding transaction $tx_{A,B}$ has been confirmed on the blockchain. From this point, A and B can have off-chain transactions via updating channel states which assign each of a new balance as $\texttt{UpdateChannel}(\gamma_{A,B}, \alpha+\beta, bal_A, bal_B)$ (i.e., A and B jointly sign a new channel state $tx'_{A,B}$ with outputs $\alpha' = bal_A$ and $\beta' = bal_B$). Either A or B can close $\gamma_{A,B}$ by posting the final state $tx'_{A,B}$ on the blockchain.

2) Payment channel network: However, one user cannot maintain one PC with each user because of the deposits, thereby making the PC technique impossible to achieve the desired effect. PCNs continue to build on PCs to enable any two users to be connected through a path of intermediate PCs. For example, if users A and B both have a PC with user I, then I can be the intermediary to support the payments between A and B (i.e., A pays coins α to I via $\gamma_{A,I}$ and I pays coins α to B via $\gamma_{I,B}$, which is equivalent to user A pays coins α to B). Therefore, the fundamental security property *atomicity* of PCNs guarantees that either the payments happen in all channels or none at all.

The Bitcoin-based LN is the state of the art in both PC and PCNs, where *atomicity* is guaranteed mainly based on the Hashed Timelock Contracts. In the LN, all users on the path lock some coins via hashed timelocks, which can only be opened by a unique witness known to the receiver of each PC. In particular, once a receiver has opened a hashed time-lock to get the coins, then simultaneously the witness is released to the receiver of the next hop. Nevertheless, this construction brings several drawbacks: 1) low robustness: the success of each payment strongly depends on each intermediary on the path; 2) high cost: as each payment needs to pass coins through the intermediate users one by one, it highly costs not only time but also the fees; 3) weak privacy: the intermediate users know about every payment between the end users. To mitigate these issues, VCs have been proposed [18].

3) Virtual channel: The VC, built over the underlying PCNs, enables any two users to establish a direct channel. To understand the differences of payments in the PCN and VC, we use Fig. 2 to show an example. Assume users A and B want to have payments, while no direct channel between them, i.e., there exists PCs $\gamma_{A,I}$ and $\gamma_{I,B}$. Then the MHPs enables users A and B to pay each other via the intermediate user I with the drawbacks we have discussed above, in particular, user I is involved in each off-chain payment of users A and B. Instead, the VC $\widehat{\gamma}_{A,B}$ ensures users A and B securely have off-chain payments directly via the blue arrow, while the intermediate user I only needs to participate in the setup and close phases. Thus, the VC can be seen as an important building block for efficiently implementing PCNs.



(a) PCN construction (b) VC construction

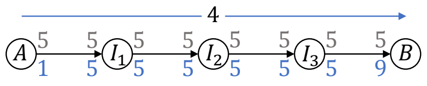
Fig. 2: Illustration of users A and B having payments through the intermediary I respectively in the PCN and VC.

Unsurprisingly, it is challenging to implement the VC in actual designs. Previous works [17, 19, 26] show how to construct multi-hop VCs with smart contract or extra scripts as the trusted third party, which is incompatible with most existing blockchains like [5]. The following work Donner [10] successfully removes the trusted third party with the trusted assumption that intermediate users are always online, which ensures that the users can respond promptly to secure their coins against malicious activities. Nonetheless, the assumption of perpetual online connectivity is often impractical, and any periods of absence could be exploited by malicious actors.

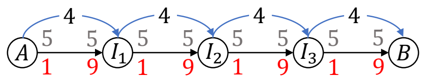
C. Balance skewness in MHPs

Firstly, it is crucial to highlight that the capacity of a MHP is constrained by the minimum balance along the path. Besides, we informally define the sustainability of PCNs as follows.
Sustainability. A PCN achieves sustainability if its MHP protocols not only meet current payment demands but also do not impair the subsequent payment capabilities in both directions of the involved channels.

In a typical PC, the flow of coins in both directions is often imbalanced and gradually accumulates toward one user. Consequently, the channel depletes in that direction, ultimately becoming unidirectional or closing altogether, resulting in subsequent multi-hop payments failing. Worse still, this issue exacerbates routing difficulties and increases routing costs, further compromising the availability of networks. Channel depletion stands as the paramount challenge in efficiently realizing both payments and PCNs. However, existing MHP protocols themselves mainly contribute to this phenomenon, as described in more detail below:



(a) Ideal multi-hop payment



(b) Real multi-hop payment

Fig. 3: Diagrams of balance variation within a multi-hop payment, where (a) is the ideal case, (b) is the real case.

Ideally, as depicted in Fig. 3(a), after the MHP between users A and B (i.e., user A pays coins 4 to B), the balances of the intermediate users I_1, I_2 and I_3 in each channel should be unchanged (here we do not discuss the fees paid to the intermediaries). However, in the real MHP protocol (Fig. 3 (b)), the payment of coins 4 occurs in each channel and thus results in fund depletion for each underlying PC (i.e., each intermediary I owns balance 1 in the left channel while 9 in the right channel), thereby restricting the payment capabilities of these channels. This, in turn, undermines the sustainability of PCNs. Furthermore, this will weaken the enthusiasm of users to participate in MHPs, as they not only provide a service (including locking up part of their assets for a long period) but also face the risk of fund depletion and even being forced to close the channel. Notice that, all channels along the path suffer from the risk of each MHP, leading to numerous channels being depleted in PCNs. Designing a rebalancing protocol to address this issue and fundamentally avoid channel depletion, thereby enhancing the sustainability of PCNs, remains an open challenge.

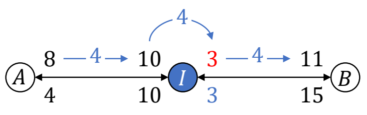
III. Solution overview

In this work, our ultimate goal is to securely realize the ideal MHPs by proposing an efficient rebalancing protocol that avoids balance skewness, ensuring successful payments as long as each intermediary has sufficient balance in adjacent channels (i.e., it can only be rich in one of these two channels). This section begins with a high-level overview of our approach, followed by outlining the basic construction in a three-party scenario. Subsequently, we delve into the challenges and explore potential solutions, eventually leading to the final protocol. The protocol is formally and thoroughly described in Section IV-C.

A. *Flow neutrality*

Considering the major causes of channel depletion, i.e., the balance skewness in each channel during the MHPs, we propose a new approach named *flow neutrality*, where the *flow* implies the directed payments and *neutrality* implies offsetting the positive and the negative.

Flow neutrality. Instead of conducting payments within each channel individually, intermediary users immediately utilize funds received from one channel for the subsequent payment in the next channel, thereby offsetting the impact on their balances. In essence, *flow neutrality* aims to minimize the impact on channel balance allocations during MHPs. As a result, once the *flow neutrality* is achieved across the current PCNs, the risk of channel depletion will be significantly reduced, enhancing the sustainability of PCNs and further improving their scalability and throughput.



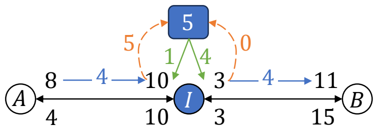
We first consider achieving *flow neutrality* in a three-party structure, where the intermediate user opens two channels with different users, as illustrated above. User I maintains channel $\gamma_{A,I}$ with user A and also channel $\gamma_{I,B}$ with user B, where I owns balance 10 in $\gamma_{A,I}$ but 3 in $\gamma_{I,B}$. Suppose now A needs to make a payment of 4 coins to B, which will fail to go through I.

An intuitive but imperfect construction. A natural idea is to transfer the 4 coins in $\gamma_{A,I}$ to $\gamma_{I,B}$ and complete the payment (ignoring fees for simplicity). For user I, the balances in the two channels will remain unchanged, satisfying *flow neutrality*. In other words, regardless of how many transactions occur between A and B, I will maintain balance equilibrium, keeping a stable amount of funds in both $\gamma_{A,I}$ and $\gamma_{I,B}$. For other users who need to utilize $\gamma_{A,I}$ and $\gamma_{I,B}$ channels as payment intermediaries, their payment paths will also remain stable and well-funded.

The aforementioned MHP construction rebalances and mitigates the balance skewness introduced by unbalanced payments in the underlying channels. Besides, it amplifies the payment capabilities of intermediate users. However, the on-chain claim required for fund transfers brings additional costs and a potential privacy concern. For instance, adversaries might gain extra information by observing the fund transfers caused by MHPs. Additionally, from a performance perspective, a more efficient scheme for implementing MHPs is needed. At this point, our goal is to construct a VC over the PCs between the two end users, while retaining the design for rebalancing introduced in the previous section.

B. *Three-party VC construction*

We still begin by considering VC construction in a three-party scenario. The basic logic is that the opening operation implies transferring coins from channels to the VC while the closing operation implies returning coins from the VC to underlying channels, as shown in the diagram below. In slightly more detail, intermediary I first shifts 5 coins of $\gamma_{A,I}$ to an account jointly controlled by A, I, and B for supporting the payments between A and B (represented by the yellow dashed line in the diagram). After the claim has been confirmed, i.e., the VC $\widehat{\gamma}_{A,B}$ between A and B is successfully established, A and B can have unlimited bi-directional off-chain payments privately with upper bound 5, without involving I. When A and B decide to finish with the result that A should pay 4 coins to B, then A, B, and I move to the closing phase, as represented by the green solid line in the diagram.



In the closing phase, the 5 coins are split into two parts 1 and 4 for $\gamma_{A,I}$ and $\gamma_{I,B}$ respectively. Before withdrawing the 5 coins, $\gamma_{A,I}$ and $\gamma_{I,B}$ are updated according to the allocation of these 5 coins. Once the 5 coins are returned to the underlying channels, the VC $\widehat{\gamma}_{A,B}$ is closed, and the balances of user I in channels $\gamma_{A,I}$ and $\gamma_{B,I}$ remain at 10 and 3, respectively. Thus, the insufficient balance of I no longer prevents $\gamma_{I,B}$ from participating in MHPs. In addition, the underlying channels $\gamma_{A,I}$ and $\gamma_{I,B}$ can still work normally while the VC $\widehat{\gamma}_{A,B}$ works. The VC $\widehat{\gamma}_{A,B}$ not only brings less cost and better efficiency of trades for A and B but also amplifies I's payment capabilities in both directions without negative changes on the balances in either channel. Next, we will explain why our VC construction is resistant to the *domino attack*.

Resist domino attack. As Donner [10] describes, the *domino attack* targets VCs constructed as proposed in work [8] and the recursive ones, e.g., Elmo [26] and LVPC [24]. The direct effect of *domino attack* is to force the closure of all the underlying PCs on this path and the quantitative experiment shows that a malicious user would force all existing channels in LN to close at a very low cost. In work [8], to protect the balance security of users, it provides the offloading process to claim their coins on-chain. However, both end users make their funding transactions off-chain, so the only way to offload is to close their channels with on-chain transactions. That is the core reason that *domino attack* works. In this work, we separate the offloading process of VCs from closing underlying channels, since the channel capacity is funded to a public account controlled by the three parties. When a dispute occurs, it is only necessary to allocate the funds from the public account back to the underlying channels to offload the VC. Therefore, our construction inherently resists *domino attack*.

Intuition on sustainability. In our approach, intermediate users achieve coin transfers between adjacent channels by redistributing funds from the three-party account to the two channels upon closing the VC. This naturally offsets the balance skewness caused by MHPs, thereby remaining the payment capabilities of channels. Consequently, our approach will endow PCNs with robust *sustainability* and performance.

Actually, when the involved users are more than 3, the above process is not secure enough to guarantee *atomicity* that honest users do not lose coins and further measures are needed (see next subsection).

C. Multi-party VC construction

Let us further consider a more complex situation where users U_0 and U_n trade with each other through the intermediaries $U_i(i \in [1, n-1])$ on the path ($U_0 \leftrightarrow U_1 \leftrightarrow \cdots \leftrightarrow U_{n-1} \leftrightarrow U_n$) via opening a VC $\widehat{\gamma}_{0,n}$ with capacity α and lifespan T. Based on the three-party construction in the previous section, we extend it to support MHPs as shown in Fig. 4.

One of the core differences between MHPs and VCs is that each MHP is publicly available to all participating users, while each payment within the VC is only known to the end users, with other users only participating in the final closing phase. Therefore, there is one multi-channel update needed when the VC is about to close. Just as we know, one of the most vital challenges in constructing the VC is to close it safely, which is decided by the honest completion of the multi-channel update for the final payment.

For the three-party VC construction, achieving honest and consistent multi-channel updates is much easier. The intermediate user I is able to coordinate channel updates on both sides, ensuring update consistency under the supervision of end users A and B. However, when extended to multi-party construction, two challenges urgently need to be addressed: 1) intermediate users cannot tell if the updates are consistent with the latest VC state; 2) end users cannot supervise the intermediate users. To address these two specific issues, it is necessary to ensure that all users will choose the latest state to update their channels. We first observe that there is no mutual trust between the sender and receiver. When one of them selects an outdated state to close the VC, the other one must lose money. Thus if the sender and receiver compete to provide the state for closing the VC, where the state with larger timestamp will be selected, we are confident that the rational sender and receiver will agree on the latest VC state to keep their assets from damaging. Considering the lack of reliance on trusted third parties and scripts, who would be the referee? As the intermediate users $U_i(i \in [1, n-1])$ also need to know how to update in the close phase, we introduce an *incentive time-based fee model* that favors the latest state for the intermediate users and assigns this responsibility to them.

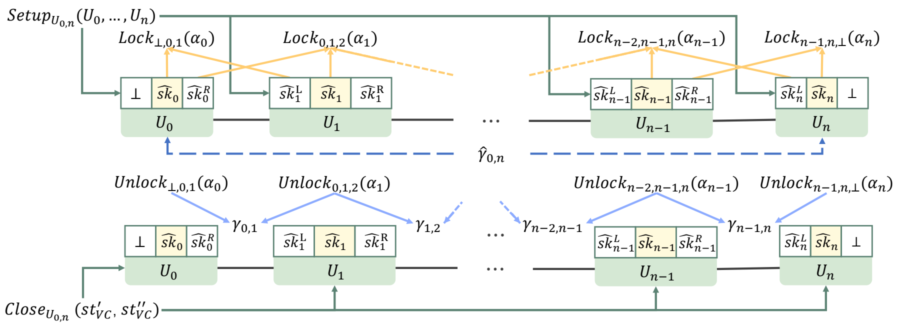
Incentive fee model based on time. We incorporate timestamps into VC transactions to establish the sequence of events during the update phase. When parties opt to close the VC, each submits a credential reflecting their chosen VC state to the intermediate users. As mentioned above, there will be one rational $P(P \in U_0, U_n)$ chooses the latest state st' with the α' to pay and the timestamp T'. Then the intermediate users $U_i(i \in [1, n-1])$ calculate their fees by $f' = fee(T')$, where $fee(t) = (\frac{t}{T})^2 \cdot f$. Here, f represents the total fee for the whole lifetime T, and t represents the time at the latest state. Due to the curvilinear nature of the power function, it can incentivize intermediate users to perform services for a longer time and motivate them to choose the state with the most recent timestamp in the competition for end users.

When a consistent state has been determined, the next step is to perform updates of all the underlying channels. For security purposes, previous works such as Shaduf [21], have required users to stay online to ensure a timely response to malicious behaviors. Also, for efficiency, users expect the closing phase to be swift, freeing their locked assets quickly, which necessitates that all users update their channels in a timely manner. However, the online assumption is unrealistic. To remove this assumption, we propose a timing mechanism named *Time Slicing* to a pseudo-real-time effect without requiring the users to be online in real-time, while ensuring both efficiency and safety.

Time Slicing. The timing mechanism splits the entire lifetime T of the VC to smaller time slices $t_i(i \in [0, n])$, during which the intermediate users are online, awaiting closing requests from the end users and completing the close phase. The interval time φ between two adjacent slices is designated as the safe offline period for users until the next time slice. In the implementations, the lifetime T is set to a large value (e.g., a few days) to ensure enough time for completing the payments between end users U_0 and U_n, the exact values of t and φ are based on the network delay and level of congestion on the blockchain, and φ is substantially longer than t. For instance, in a real-case scenario, such as Horcrux on Bitcoin with a VC lifespan T of 48 hours, each time slice t can be set to 10 minutes, with a time interval φ of 7 hours and 50 minutes between slices for users to securely go offline. This divides the lifespan T into 6 time slices, requiring the intermediate users to be online for only 1 hour in total, significantly reducing their burden compared to being online for the full 48 hours. [1]

As previously outlined, the widespread reuse of identical payment paths results in channels progressively leaning in one direction or even reaching depletion. Consequently, users actively involved in MHPs are particularly susceptible, thereby limiting the extensive utilization of related payment paths. Our solution plays a crucial role in mitigating the risk of depletion for these active users, ultimately ensuring the maintenance of a balanced and sustainable PCN. For two users with multiple transaction demands but without a directional channel between them, our solution enables the cost-effective execution of multiple bidirectional transactions in a single multi-channel update. Simultaneously, this process rebalances the intermediary channels while amplifying the payment capacity of each user to their overall balance in the two adjacent PCs. This

[1] Clearly, *Time Slicing* not only weakens the strong real-time online assumption, but also guarantees the timely and secure closure of Horcrux (cf. Section IV-C for detailed analysis).



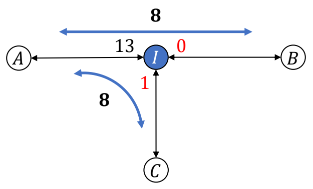
$Setup_{U_{0,n}}(U_0,\ldots,U_n)$
$Lock_{\perp,0,1}(\alpha_0)$
$Lock_{0,1,2}(\alpha_1)$
$Lock_{n-2,n-1,n}(\alpha_{n-1})$
$Lock_{n-1,n,\perp}(\alpha_n)$
U_0
U_1
U_{n-1}
U_n
$\hat{\gamma}_{0,n}$
$Unlock_{\perp,0,1}(\alpha_0)$
$Unlock_{0,1,2}(\alpha_1)$
$Unlock_{n-2,n-1,n}(\alpha_{n-1})$
$Unlock_{n-1,n,\perp}(\alpha_n)$
$\gamma_{0,1}$
$\gamma_{1,2}$
$\gamma_{n-2,n-1}$
$\gamma_{n-1,n}$
$Close_{U_{0,n}}(st'_{VC}, st''_{VC})$

Fig. 4: Two multi-party updates of underlying channels respectively in Setup phase and Close phase

results in a mutually beneficial scenario for both the end users and intermediary users.

Necessity of on-chain confirmation. While transferring coins between adjacent channels simultaneously with MHPs can effectively avoid introducing balance skewness, it also changes the funds reserved in the two channels. To ensure these transfers take effect and can be verified by other users, the coin transfers need to be declared on the blockchain. Otherwise, it could be vulnerable to the *double-shifting attack* [21].

Consider a scenario where the intermediate user I maintains channels $\gamma_{A,I}$, $\gamma_{I,B}$, and $\gamma_{I,C}$ with users A, B, and C respectively, with balances of 13, 0, and 1. If transferring coins from one channel to another is declared off-chain (i.e., without on-chain confirmation), malicious users A and I can collaborate to deceive honest users B and C. Specifically, in a concurrent trade with both users B and C through intermediate user I who claims to shift 8 coins to $\gamma_{I,B}$ and $\gamma_{I,C}$ respectively. As a result, in the respective views of users B and C, both fund transfers are successful. However, user I owns only 14 coins in total, so it cannot pay 16 coins in the end, resulting in a loss of 2 coins for either honest user B or C.



The core reasons that lead to *double-shifting attack* are: 1) the operations of transferring coins change the capacity of the corresponding channels; 2) balance allocation within one channel is private and obscure. Therefore, similar to other works in this area (such as Shaduf [21]), the on-chain claim is essential for protecting honest users from information asymmetry, ensuring that cross-channel coin transfers can be effectively executed.

IV. Formal definitions of Horcrux

Notations. We denote by λ the security parameter and $A(x;r) \rightarrow y$ or $y \leftarrow A(x;r)$ the output of algorithm A with inputs $x \in \{0,1\}^\lambda$ and randomness $r \in_\$ \{0,1\}^\lambda$ (r is only mentioned explicitly when required). The item of VC is specified with notation $\widehat{\ }$, and the one with superscript $\{fud, c, R, L\}$ is respectively involved in the *funding*, *closing*, *right* side and *left* side operation, and the one with subscript $1,\cdots,\ell$ is involved in the users $U_1,\cdots,U_\ell$.

A. The security model

We model the security of *Virtual Channel* in the Universal Composability (UC) framework [16] and deploy the version with a global setup (GUC) [15]. We define the model over the set of participating users $U_0,\cdots,U_n$, where users U_0 and U_n have bidirectional payments via the intermediate users $\{U_1,\cdots,U_{n-1}\}$, and take the underlying blockchain as $\mathbb{B}$ as a global ideal functionality $\mathcal{F}_\mathbb{B}$ with the maximum confirmation delay Δ. The GUC model specifies two worlds, a real protocol Π is executed in the real world by the users, interacting with the adversary $\mathcal{A}$ and environment $\mathcal{Z}$; while, in the ideal world, an ideal functionality $\mathcal{F}$ is executed by the users, interacting with the simulator $\mathcal{S}$ and environment $\mathcal{Z}$. Specifically, we denote the ensemble corresponding to the real-world protocol execution as $EXEC^{\mathcal{F}_\mathbb{B}}_{\Pi,\mathcal{Z},\mathcal{A}}(\lambda)$ and the ensemble corresponding to the ideal world execution as $EXEC^{\mathcal{F}_\mathbb{B}}_{\mathcal{F},\mathcal{Z},\mathcal{S}}(\lambda)$.

The adversary model. Throughout the paper, we consider the rational adversary $\mathcal{A}$ who seeks to maximize profit. In addition, the static adversary $\mathcal{A}$ chooses any users in $\{U_0,\cdots,U_n\}$ to fully control before the protocol starts and only ensures that at least one of $\{U_0,U_n\}$ is honest.

Communication network. We assume the round synchronous communication network and each user is aware of the current round number. In slightly more detail, the message is delivered within one round, i.e., the message m sent at round r will be received by all the users at the beginning of round $r+1$. We also assume that authenticated communication channels exist between any two users. Additionally, the adversary $\mathcal{A}$ can see the delivered message but cannot modify or drop it.

The blockchain ideal functionality. Similar to previous work [21], we consider the underlying blockchain as a global ledger functionality $\mathcal{F}_\mathbb{B}$ parameterized by a signature scheme $SIG \in \{Schnorr, ECDSA, BLS\}$ (cf. Fig. 5). Informally, $\mathcal{F}_\mathbb{B}$ tracks the current balance $pk.bal \in \mathbb{N}$ of each account, identified by a public key pk. Balances can be spent using the corresponding secret key sk, where $(sk,pk) \in SIG.KGen(\lambda)$. Any user can invoke interface Transfer$[\{(pk_{s_1},v_{s_1}),(pk_{s_2},v_{s_2}),\cdots\},\{(pk_{r_1},v_{r_1}),(pk_{r_2},v_{r_2}),\cdots$

$\}, \{sk_{s_1}, sk_{s_2}, \cdots\}]$ to transfer coins $v_{s_1}, v_{s_2}, \cdots$ respectively from sending accounts $pk_{s_1}, pk_{s_2}, \cdots$ to the receiving accounts $pk_{r_1}, pk_{r_2}, \cdots$ using the corresponding secrets keys $sk_{s_1}, sk_{s_2}, \cdots$, where $v_{s_1} + v_{s_2} + \cdots = v_{r_1} + v_{r_2} + \cdots$.

The security definition. Protocol Π UC-realizes the ideal functionality $\mathcal{F}$ with respect to the global blockchain functionality $\mathcal{F}_{\mathbb{B}}$, if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that $EXEC^{\mathcal{F}_{\mathbb{B}}}_{\Pi,\mathcal{Z},\mathcal{A}} \approx EXEC^{\mathcal{F}_{\mathbb{B}}}_{\mathcal{F},\mathcal{Z},\mathcal{S}}$, where $\approx$ denotes the computational indistinguishability.

Remark. Throughout the paper, we strictly separate the *parties* who are responsible for maintaining the secure execution of the underlying blockchain and the *users* who only use the service provided by the underlying blockchain (i.e., posting transactions on the blockchain).

The functionality $\mathcal{F}_{\mathbb{B}}$ interacts with users $U_0, U_1, \cdots, U_n$, the environment $\mathcal{Z}$, and simulator $\mathcal{S}$. It is parameterized by a signature scheme $SIG := (KGen, Sign, Vrf)$.

Initialization. Upon receiving the account-balance pair (pk, v) from the environment $\mathcal{Z}$:

```
01 It sets $pk.bal = v$.
02 It sets the public ledger as $\mathcal{L} := \{(pk_1, v_1), (pk_2, v_2), \cdots, (pk_\ell, v_\ell)\} \in \mathbb{R}^{2\ell}_{\geq 0}$ and sends $\mathcal{L}$ to $\mathcal{S}$.
```

Transfer. Transfer$[\{(pk_{s_1}, v_{s_1}), (pk_{s_2}, v_{s_2}), \cdots\}, \{(pk_{r_1}, v_{r_1}), (pk_{r_2}, v_{r_2}), \cdots\}, \{sk_{s_1}, sk_{s_2}, \cdots\}]$ called by user U_i:

```
01 If $(v_{s_1} > pk_{s_1}.bal) \vee (v_{s_2} > pk_{s_2}.bal) \vee \cdots$, then aborts.
02 If $v_{r_1} + v_{r_2} + \cdots > v_{s_1} + v_{s_2} + \cdots$, then aborts.
03 If $(pk_{s_1}, sk_{s_1}) \notin SIG.KGen(\lambda) \vee (pk_{s_2}, sk_{s_2}) \notin SIG.KGen(\lambda) \vee \cdots$, then aborts.
04 It sets $pk_{s_1}.bal := pk_{s_1}.bal - v_{s_1}, \cdots$ and $pk_{r_1}.bal := pk_{r_1}.bal + v_{r_1}, \cdots$.
05 It updates ledger $\mathcal{L}$ and sends it to $\mathcal{S}$.
```

Fig. 5: The Blockchain Ideal Functionality $\mathcal{F}_{\mathcal{B}}$

B. Ideal functionality of virtual channel

To model the security of the multi-party VC in our setting, we require the ideal functionality guarantees *atomicity*: for each hop, the honest receiver cannot lose coins. Formally, the ideal functionality $\mathcal{F}$ (see Fig. 6) communicates with the users U_i $(i \in [0, n])$, the environment $\mathcal{Z}$, the simulator $\mathcal{S}$ and the underlying blockchain $\mathcal{F}_{\mathbb{B}}$. Ideal functionality $\mathcal{F}$ consists of three phases and each one is triggered by the message sent by user U_i $(i \in [0, n])$.

(A) Setup Phase-Freezing Channel Coins. This phase enables the users U_0 and U_n to open a VC $\widehat{\gamma}_{0,n}$ with capacity α and lifetime T. In slightly more details, along the path between users U_0 to U_n (denoted by $U_0 \leftrightarrow U_1 \leftrightarrow \cdots \leftrightarrow U_{n-1} \leftrightarrow U_n$), each user U_i $(i \in [0, n])$ agrees with the opening of $\widehat{\gamma}_{0,n}$ by sending *Open* message $(\texttt{Open}, \alpha_i^L, \alpha_i^R, sk_i^L, sk_i^R)$ to ideal functionality $\mathcal{F}$, which specifies the coins α_i^L in channel $\gamma_{i-1,i}$ and coins α_i^R in channel $\gamma_{i,i+1}$ are to be frozen. Ideal functionality $\mathcal{F}$ updates each user U_i's balance bal_i^L and bal_i^R in channels $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$ respectively, and invokes the subroutine Freeze $[\{(pk_{i-1,i}, \alpha_i^L), (pk_{i,i+1}, \alpha_i^R)\}, pk_{\mathcal{F},i}, (sk_{i-1}^R, sk_i^L), (sk_i^R, sk_{i+1}^L)]$ to transfer coins α_i^L and α_i^R respectively from addresses $pk_{i-1,i}$ and $pk_{i,i+1}$ to a specific address $pk_{\mathcal{F},i}$ controlled by $\mathcal{F}$, where $\alpha_i^L + \alpha_i^R \geq \alpha + Max\{i, n-i\} \cdot fee(T)$.

(B) Payment Complete Phase. At time $t_k < T$ $(k \in \mathbb{Z}^+)$, user $U_i (i \in \{0, n\})$ initiates a payment of value $\overline{\alpha}_k$ to U_{n-i}. The ideal functionality $\mathcal{F}$ computes the overall coins $\widehat{\alpha}_k$ that user U_0 pays to U_n and, if $|\overline{\alpha}_k| \leq \alpha$, it updates record $(\widehat{\alpha}, t) := (\overline{\alpha}_k, t_k)$, where $(\widehat{\alpha}, t)$ is initiated as ϕ.

(C) Payment Close Phase. The frozen coins are released to the corresponding PCs. Here we consider two conditions: (1) no payment occurs between users U_0 and U_n during time T, then the frozen coins α_i^L and α_i^R are unlocked to their original PCs, (2) user U_0 has paid coins $\widehat{\alpha}$ to U_n until time $t < T$, for the receiver U_i of each hop, if it has frozen enough coins in this PC (i.e., $\alpha_i^L \geq \widehat{\alpha} + (n-i)fee(t)$ for $\widehat{\alpha} \geq 0$ and $\alpha_i^R \geq -\widehat{\alpha} + ifee(t)$ for $\widehat{\alpha} < 0$), then the frozen coins $\alpha_i^R + \widehat{\alpha} + (n-i)fee(t)$ for $\widehat{\alpha} \geq 0$ or $\alpha_i^L - \widehat{\alpha} + ifee(t)$ for $\widehat{\alpha} < 0$ are unlocked to the payment side; otherwise, the all the frozen coins $\alpha_i^L + \alpha_i^R$ are unlocked to the payment side.

Now we analyze that the ideal functionality $\mathcal{F}$ guarantees the *atomicity*:

- *Successful Payment*: If VC $\widehat{\gamma}_{0,n}$ is opened successfully via respectively transferring the involved users' coins from channels to the accounts controlled by $\mathcal{F}$, then $\mathcal{F}$ will record each payment between users U_0 and U_n as $(\widehat{\alpha}, t)$ during the Payment Complete Phase; and in the Payment Close Phase, $\mathcal{F}$ securely close the VC $\widehat{\gamma}_{0,n}$ via respectively unfreezing the coins (controlled by $\mathcal{F}$) to corresponding PCs according to local record $(\widehat{\alpha}, t)$. Especially, before successfully closing VC $\widehat{\gamma}_{0,n}$, the underlying PCs are updated accordingly to ensure the unfrozen coins are indeed returned to their intended receivers.
- *Failed Payment*: If there exists the adversarial user along the path failing to initiate its *Open* operation until timeout T, then $\mathcal{F}$ will unlock the frozen coins to their original PCs (i.e., $(\widehat{\alpha}, t) = \phi$).

C. Detailed construction

Recall in the setting, users U_0 and U_n, connected via PCN $U_0 \leftrightarrow U_1 \leftrightarrow \cdots \leftrightarrow U_{n-1} \leftrightarrow U_n$, wish to have bidirectional payments with upper bound $\alpha > 0$. Specifically, user U_i with its left and right neighbors U_{i-1} and U_{i+1} jointly control PCs $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$ respectively. And user U_i respectively owns balance bal_i^L and bal_i^R in channel $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$, where $bal_i^L + bal_i^R \geq \alpha + Max\{i, n-i\} \cdot fee(T)$. In addition, we set lifespan T as the maximum tolerable duration of locking coins in the three-party account, and if $i = 0$ then $i - 1 = \perp$, and if $i = n$ then $i + 1 = \perp$.

Protocol details. We provide a detailed description of Horcrux in Fig. 7, and the key ideas of each phase is presented below:

(A) Setup Phase-Freezing channel coins into a three-party account. This phase allows each user U_i $(i \in [0, n])$ to transfer coins a_i^L and a_i^R respectively from the channels $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$ to a three-party account $\widehat{pk}_{i-1,i,i+1}$ via a funding transaction tx_i^{fud}, where $bal_i^L + bal_i^R > a_i^L + a_i^R > \alpha + Max\{i, n-i\} \cdot fee(T)$, and the three-party account $\widehat{pk}_{i-1,i,i+1}$ is jointly controlled by users U_{i-1}, U_i and U_{i+1}. Importantly, to guarantee the secure establishment of VC $\widehat{\gamma}_{0,n}$, before posting the transaction tx_i^{fud} on blockchain, user U_i updates the underlying PCs $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$ with

(A) Setup Phase-Freezing Channel Coins

01 Upon receiving $(\texttt{Open},\widehat{\gamma}_{0,n},\gamma_{0,1},sk_0^R,U_0,U_n,\alpha,T)\overset{r_1}{\hookleftarrow}U_0$ and $(\texttt{Open},\widehat{\gamma}_{0,n},\gamma_{n-1,n},sk_n^L,U_n,U_0,\alpha,T)\overset{r_1}{\hookleftarrow}U_n$, it sends $(\texttt{OpeningVirtual Channel},\gamma_{i-1,i},\gamma_{i,i+1},\alpha,T)\overset{r_1+1}{\hookrightarrow}U_i\ (i\in[1,n-1])$. // Opening VC $\widehat{\gamma}_{0,n}$ with value α and lifetime T between users U_0 and U_n.

02 Upon receiving $(\texttt{Open},\alpha_i^L,\alpha_i^R,sk_i^L,sk_i^R)\overset{r_1+2}{\hookleftarrow}U_i\ (i\in[1,n-1])$, for each $U_i\ (i\in[0,n])$, it sends $(\texttt{ChannelUpdate}, bal_i^L := bal_i^L-\alpha_i^L, bal_i^R := bal_i^R-\alpha_i^R)\overset{r_1+3}{\hookrightarrow}U_i$. If $(\texttt{ChannelUpdate},ok)\overset{r_1+4}{\hookleftarrow}U_i$, it invokes subroutine Freeze $[\{(pk_{i-1,i},\alpha_i^L),(pk_{i,i+1},\alpha_i^R)\},\{pk_{\mathcal{F},i}\},\{(sk_{i-1}^R,sk_i^L),(sk_i^R,sk_{i+1}^L)\}]$ and sends $(\texttt{Opened})\overset{r_1+5}{\hookrightarrow}U_i$. Otherwise, aborts. // Transferring coins from PC to account $pk_{\mathcal{F}}$ controlled by functionality $\mathcal{F}$.

(B) Payment Complete Phase

01 Upon receiving $(\texttt{Payment},\widehat{\gamma}_{0,1},\overline{\alpha}_k)\overset{r_2>r_1+5}{\hookleftarrow}U_i\in\{U_0,U_n\}$, it sets $\widehat{\alpha}_k:=\widehat{\alpha}_{k-1}+\overline{\alpha}_k$ for $i=0$ and $\widehat{\alpha}_k:=\widehat{\alpha}_{k-1}-\overline{\alpha}_k$ for $i=n$. If $|\widehat{\alpha}_k|\le\alpha$, then it sends $(\texttt{VirtualChannelUpdate}, U_i\to U_{n-i},\overline{\alpha}_k,t_k,\widehat{\alpha}_k)\overset{r_2+1}{\hookrightarrow}U_i\in\{U_0,U_n\}$; otherwise, aborts. // User U_i pays coins $\overline{\alpha}_k$ to U_{n-i}.

02 Upon receiving $(\texttt{VirtualChannelUpdate},ok)\overset{r_2+2}{\hookleftarrow}U_i\ (i\in\{0,n\})$, it sends $(\texttt{Paid})\overset{r_2+3}{\hookrightarrow}U_i\ (i\in\{0,n\})$ and updates value-time pair as $(\widehat{\alpha},t):=(\widehat{\alpha}_k,t_k)$.

(C) Payment Close Phase

01 At time T, if $(\widehat{\alpha},t)=\phi$, // No payment occurs between users U_0 and U_n.

- For each $U_i\ (i\in[0,n])$, it sends $(\texttt{ChannelUpdate}, bal_i^L := bal_i^L+\alpha_i^L, bal_i^R := bal_i^R+\alpha_i^R)\overset{r_3>T}{\hookrightarrow}U_i$. If $(\texttt{ChannelUpdate},ok)\overset{r_3+1}{\hookleftarrow}U_i$, it invokes subroutine Unfreeze $[\{pk_{\mathcal{F},i},\alpha\},\{(pk_{i-1,i},\alpha_i^L),(pk_{i,i+1},\alpha_i^R)\},\{sk_{\mathcal{F},i}\}]$ and sends $(\texttt{Closed},\widehat{\gamma}_{0,n})\overset{r_3+2}{\hookrightarrow}U_i$.

02 Upon receiving $(\texttt{Close},\widehat{\gamma}_{0,n})\overset{r_4>r_2+3}{\hookleftarrow}U_i\ (i\in\{0,n\})$ at time $t<T$, it sends $(\texttt{Closing},\widehat{\gamma}_{0,n})\overset{r_4+1}{\hookrightarrow}U_{n-i}$. // User U_0 paid coins $\widehat{\alpha}$ to U_n.

- For each user $U_i\ (i\in[0,n])$, it respectively updates U_i's balance in channels $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$ by invoking subroutine Update$[\widehat{\alpha},\gamma_{i-1,i},\gamma_{i,i+1}]$.
- Upon receiving $(\texttt{ChannelUpdate},ok)\overset{r_4+3}{\hookleftarrow}U_i$, it unfreezes the coins in $pk_{\mathcal{F},i}$ to channels $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$ respectively by invoking subroutine Close$[(pk_{\mathcal{F},i},sk_{\mathcal{F},i}),\widehat{\alpha},\gamma_{i-1,i},\gamma_{i,i+1}]$.

* The above subroutines are in Fig. 10 of Appendix B.

Fig. 6: The Ideal Functionality $\mathcal{F}$

neighbors U_{i-1} and U_{i+1} respectively, i.e., subtracting coins α_i^L from balance bal_i^L and coins α_i^R form balance bal_i^R.

(B) Payment Complete Phase. This phase enables the users U_0 and U_n to freely have payments without interacting with the intermediaries. In particular, both users U_0 and U_n can initiate a payment and for the $k+1$-th payment, they first compute the relative amount of coins that U_0 should pay to U_n as $\widehat{\alpha}_{k+1}\in\mathbb{N}$ and then respectively sign the latest VC state st_{vc}. Meanwhile, we use monotonically increasing sequence $\{t_1,\cdots,t_k,\cdots\}$ to denote the specific time of each payment.

(C) Payment Close Phase. From the aspect of the rational intermediate users, they wish the VC $\widehat{\gamma}_{0,n}$ can be closed at time T such that they can obtain the maximum transaction fee $fee(T)$. Thus, we consider the following two conditions: 1. If no payment occurs until time T, then each user $U_i\ (i\in[0,n])$ cooperates with its left and right neighbors to return the coins in three-party account $\widehat{pk}_{i-1,i,i+1}$ to the original PCs $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$; 2. Either user U_0 or U_n can initiate the *close* operation via sending the latest VC state st_{vc} to all the intermediate users before time T. According to the latest VC state st_{vc} received from U_0 or U_n (i.e., the one with the bigger time t and signed correctly by users U_0 and U_n), each user $U_i\ (i\in[0,n])$ cooperates with its left and right neighbors to update the underlying PCs' states (i.e., computing the number of coins in three-party account $\widehat{pk}_{i-1,i,i+1}$ that will return to $\gamma_{i-1,i}$ and $\gamma_{i,i+1}$), and generate the corresponding close transaction tx_i^c to close the three-party account $\widehat{pk}_{i-1,i,i+1}$.

On the waiting time for closing Horcrux. Before delving into the security analysis of Horcrux, let us discuss the timely and secure closure of Horcrux. In a round synchronous communication network, we can reliably expect to receive the closing request from the honest end user U_0 or U_n within one round, even if he is the payer in the whole bidirectional payments. Moreover, to maximize profit, rational intermediate users will actively close VC with the latest state (i.e., the honest closing request). Therefore, upon receiving a closing request, each intermediate user only needs to wait for another round to ensure that they have received the latest VC state from the honest end user before securely closing VC with their left and right neighbors. Overall, Horcrux will take at most $\varphi+t+\Delta$ time to be finally closed, where t is long enough for completing the Payment Close Phase.

Security intuitions. We brief the security intuitions in the following and defer the detailed proofs to Appendix C. Essentially, the rational users can only maximize profits if they honestly follow the protocol specifications. Firstly, VC can only be updated jointly by the two end users, with at least one being honest. Furthermore, during the Payment Close Phase, all intermediate users are ensured to receive the correct and latest VC state within one round, and our time-based incentive mechanism guarantees all intermediate users always close VC with this latest state. More specifically:

• *Successful Payment*: This directly stems from the security of the underlying blockchain and the cryptographic building blocks (i.e., signature schemes), and our incentive mechanism (i.e., each intermediary prefers to close the VC almost near timeout T to maximize its fees). For each payment between users U_0 and U_n, they respectively sign and exchange the new VC state st_{vc} to record the time and overall payments that have occurred between them; otherwise, aborts. Additionally, since at least one of users U_0 and U_n is honest, during the Payment Close Phase, the intermediate users can always receive the valid and final VC state. Thus, each honest user can always receive the deserved coins; otherwise, the frozen coins of the adversary cannot be unlocked formally, which would harm the interests of rational users.

• *Failed Payment*: We consider the following possible cases:

– In the Setup Phase, the malicious users fail to post their funding transactions on the blockchain. Note that correctly updating the underlying PCs is the prerequisite of posting funding transactions, then the malicious users cannot hold these coins even if they are not transferred to the corresponding three-party account. Thus, the malicious users will actively participate in the protocol, otherwise, they will lose coins.

– In the Payment Complete Phase, if the VC is not updated correctly, then each malicious user cannot receive a valid state st_{vc}. Therefore, the honest user cannot lose coins.

– In the Payment Close Phase, if a malicious user U_i fails to cooperate to close the VC, then it will lose coins as its coins are still locked in VC (i.e., the three-party account $\widehat{pk}_{i-1,i,i+1}$).

Therefore, Horcrux runs as expected and satisfies *atomicity* that the honest users never lose coins.

Theorem 1. *(Atomicity) Assume Σ_{DS}^{SIG} is a secure digital signature scheme w.r.t. $SIG \in \{Schnorr, ECDSA, BLS\}$ and protocol $\Gamma_{MultiSign}^{SIG}$ is the UC-secure protocol for jointly computing signatures. Then protocol Horcrux running in the $\{\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{smt}\}$-hybrid world UC-realizes ideal functionality $\mathcal{F}$.*

V. Evaluation and comparison

In this section, we first provide a proof of concept implementation that creates the raw UTXO-based transactions necessary for the Horcrux protocol to demonstrate its feasibility. Additionally, to evaluate how Horcrux performs in the real PCN, we conduct a performance experiment comparing it with Revive [25] and Shaduf [21] in a simulation of the LN. The source code is available on Github [3].

Testbed. Our evaluation consists of a simulation using Python and the implementation of LN testnet. Specifically, we conduct experiments on the machine with the following configuration: CPU(Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz with 4 cores), RAM(16.0 GB) and OS(x64-based Windows). Based on Shaduf's evaluation benchmark, we generate channel connections between nodes from a snapshot of the real LN topology taken on March 31, 2021 [2]. The transaction values are generated from a processed LN real-world dataset [21].

A. Implementation

We implement Horcrux with generalized channels [7] as the underlying PC protocol while Horcrux remains compatible with Lightning channels. In this experiment, we focus on the on-chain costs of Horcrux, i.e., fees paid to miners when publishing transactions on-chain, so we provide additionally the expected cost in USD. In Bitcoin, the fees depend on the size of transactions and are always set as 5 sat/B. To avoid confusion in recent price fluctuations of BTC, we use the Sep 2020 exchange rate of 10602 USD per BTC (the same as Shaduf [21]). Additionally, we evaluate the time cost for each procedure in Horcrux to assess its complexity in Table II.

TABLE II: Communication overhead of Horcrux for the whole path (not per party) for the different operations with ECDSA/Schnorr based implementation, assuming a VC across n nodes. k is the number of payments between U_0 and U_n.

	txs	size (bytes)	on-chain costs (USD)	computation time (ms)	
				ECDSA	Schnorr
Setup	n	$303 \cdot n$	$0.16 \cdot n$	4.986	2.992
Update	0	$168 \cdot k$	0	0.998	0.997
Close	n	$303 \cdot n$	$0.16 \cdot n$	1.509	1.995

Cost of Horcrux. To set up a VC, each user needs to exchange 3 transactions with neighbors: the 303 bytes funding transaction tx^f and two off-chain transactions for updating channels, where the funding process costs 0.16 USD. In the update procedure, the end users exchange 168 bytes credentials with each other privately, which has no on-chain costs. When closing the VC, the end users send the last credential to the involved users. Subsequently, every user exchanges the 303 bytes closing transaction tx^c and two off-chain transactions to update channels, costing 0.16 USD for closing process. From Table II, the whole procedure for users costs about 0.3 USD.

Cost comparison. Horcrux totally costs $0.32 \cdot n$ USD for the whole path. When rebalancing between two adjacent channels, Shaduf costs 6 USD for binding and unbinding at least, and approximately 31 USD for deploying smart contracts, while Horcrux incurs less than 1 USD in total. In addition, Shaduf requires on-chain gas for invoking contracts for each rebalancing which is free in Horcrux. Therefore, as the path grows, the cost gap will increase further. Also, from Table II, the costliest procedure of Horcrux takes below 5 ms, indicating that all computations that can be executed in negligible time.

B. Evaluation

Existing rebalancing methods expand the throughput of PCNs by transferring funds from adjacent channels to depleted

```
(A) Setup Phase-Freezing channel coins into a three-party account
01 User $U_i$ completes Setup via running $SIG.KGen(1^\lambda) \rightarrow \{(\widehat{sk}_i^R, \widehat{pk}_i^R), (\widehat{sk}_i, \widehat{pk}_i), (\widehat{sk}_i^L, \widehat{pk}_i^L)\}$ and a 3PC protocol OpenAccount$(U_{i-1}, U_i, U_{i+1})$ with its right and left neighbors to obtain a three-party account $\widehat{pk}_{i-1,i,i+1}$.
02 User $U_i$ transfers coins from both channels $\gamma_{i,i+1}$ and $\gamma_{i-1,i}$ to address $\widehat{pk}_{i-1,i,i+1}$.
    • It generates funding transaction $tx_i^{fud}[\{(pk_{i-1,i}, \alpha_i^L), (pk_{i,i+1}, \alpha_i^R)\}, \{(\widehat{pk}_{i-1,i,i+1}, \alpha)\}]$ and sends $tx_i^{fud}$ to both neighbors $U_{i-1}$ and $U_{i+1}$.
    • Upon receiving transactions $tx_{i-1}^{fud}$ and $tx_{i+1}^{fud}$, it checks these two transactions via CheckValid$(tx^{fud})$, and aborts otherwise.
    • It obtains the signature of funding transaction $tx_i^{fud}$ by running MultiSign$(sk_{i-1}^R, sk_i^L, tx_i^{fud})$ with $U_{i-1}$ to obtain signature $\sigma_{i-1,i}^{fud}$ and running MultiSign$(sk_i^R, sk_{i+1}^L, tx_i^{fud})$ with $U_{i+1}$ to obtain signature $\sigma_{i,i+1}^{fud}$.
    • It updates channel $\gamma_{i-1,i}$'s state as UpdateChannel$(\gamma_{i-1,i}, bal_{i-1}^R + bal_i^L - \alpha_{i-1}^R - \alpha_i^L, bal_{i-1}^R - \alpha_{i-1}^R, bal_i^L - \alpha_i^L)$ and channel $\gamma_{i,i+1}$'s state as UpdateChannel$(\gamma_{i,i+1}, bal_i^R + bal_{i+1}^L - \alpha_i^R - \alpha_{i+1}^L, bal_i^R - \alpha_i^R, bal_{i+1}^L - \alpha_{i+1}^L)$ respectively with neighbors $U_{i-1}$ and $U_{i+1}$. After the UpdateChannel is successful, it posts $(tx_i^{fud}, \sigma_{i-1,i}, \sigma_{i,i+1})$ on blockchain.

(B) Payment Complete Phase
Assume the latest VC state payment is that user $U_0$ pays $\overline{\alpha}_k > 0$ to $U_n$ at time $t_k$ and the overall coins that user $U_0$ has paid to $U_n$ is $\widehat{\alpha}_k \in \mathbb{N}$ (denoted as $st_{vc} := (U_0 \overset{\overline{\alpha}_k}{\rightarrow} U_n, \widehat{\alpha}_k, t_k, \widehat{\sigma}_0, \widehat{\sigma}_n)$).
01 User $U_i(i \in \{0, n\}$ pays coins $\overline{\alpha}_{k+1} > 0$ to $U_{n-i}$ at time $t_{k+1}$ by doing the following:
    • It updates VC state as $[st_{vc}] := (U_i \overset{\overline{\alpha}_{k+1}}{\rightarrow} U_{n-i}, \widehat{\alpha}_{k+1}, t_{k+1})$, where $\widehat{\alpha}_{k+1} := \widehat{\alpha}_k + \overline{\alpha}_{k+1}$ for $i = 0$ and $\widehat{\alpha}_{k+1} := \widehat{\alpha}_k - \overline{\alpha}_{k+1}$ for $i = n$. Then it generates signature $\widehat{\sigma}_i \leftarrow SIG.Sign(\widehat{sk}_i, [st_{vc}])$ and sends $([st_{vc}], \widehat{\sigma}_i)$ to $U_{n-i}$.
    • User $U_{n-i}$ verifies that $U_i$ correctly updates its local VC state and $1 \leftarrow SIG.Vrf(\widehat{pk}_i, [st_{vc}], \widehat{\sigma}_i)$, then generates signature $\widehat{\sigma}_{n-i} \leftarrow SIG.Sign(\widehat{sk}_{n-i}, [st_{vc}])$ and sends $([st_{vc}], \widehat{\sigma}_{n-i})$ to $U_i$; otherwise, aborts.
Both users $U_0$ and $U_n$ store the latest VC state $st_{vc}$.

(C) Payment Close Phase
01 If no payment occurs between users $U_0$ and $U_n$ before time $T$, then after time $T$, for each user $U_i$:
    • It with $U_{i+1}$ updates channel $\gamma_{i,i+1}$'s state as UpdateChannel$(\gamma_{i,i+1}, bal_i'^R + bal_{i+1}'^L + \alpha_i^R + \alpha_{i+1}^L, bal_i'^R + \alpha_i^R, bal_{i+1}'^L + \alpha_{i+1}^L)$, generates close transaction $tx_i^c[\{\widehat{pk}_{i-1,i,i+1}, \alpha_i^L + \alpha_i^R\}, \{(pk_{i-1,i}, \alpha_i^L), (pk_{i,i+1}, \alpha_i^R)\}]$ and sends $tx_i^c$ to $U_{i-1}$ and $U_{i+1}$.
02 Otherwise, upon receiving VC state $st'_{vc}$ from user $U_0$ or $st''_{vc}$ from user $U_n$ or both $st'_{vc}$ and $st''_{vc}$ before time $T$, for each user $U_i$:
    • It sets the final VC state as $st_{vc} \in \{st'_{vc}, st''_{vc}\}$ and $st_{vc} := (U_i \overset{\overline{\alpha}_\mu}{\rightarrow} U_{n-i}, \widehat{\alpha}_\mu, t_\mu, \widehat{\sigma}_0, \widehat{\sigma}_n)$, where the one with the bigger time $t_\mu$ and $(SIG.Vrf(\widehat{pk}_0, [st_{vc}], \widehat{\sigma}_0) \rightarrow 1) \wedge (SIG.Vrf(\widehat{pk}_n, [st_{vc}], \widehat{\sigma}_n) \rightarrow 1)$.
    • According to $st_{vc}$, if $\widehat{\alpha}_\mu \geq 0$ then $U_i$ pays coins $\widehat{\alpha}_i^R := \widehat{\alpha}_\mu + (n-i-1)fee(t_\mu)$ to $U_{i+1}$; otherwise, $U_i$ pays coins $\widehat{\alpha}_i^L := -\widehat{\alpha}_\mu + (i-1)fee(t_\mu)$ to $U_{i-1}$. It updates the channel state as Redistribute$(\widehat{\alpha}_\mu, st_{vc})$.
    • It runs MultiSign$(\{\widehat{sk}_{i-1}^R, \widehat{sk}_i, \widehat{sk}_{i+1}^L\}, tx_i^c)$ with $U_{i-1}$ and $U_{i+1}$ to obtain signature $\widehat{\sigma}_{i-1,i,i+1}$. Then it posts $(tx_i^c, \widehat{\sigma}_{i-1,i,i+1})$ on blockchain.
```
* The above subroutines are in Fig. 11 of Appendix B.

Fig. 7: Pseudo-code of Horcrux

channels. In particular, the MHPs are constrained by the balances of each node along the path. Rebalancing alleviates this problem by funding coins in depleted hops, thereby facilitating more payments. However, the balance skewness in channels caused by MHPs merely exacerbates the channel depletion issue within PCNs. Horcrux is proposed to achieve *flow neutrality* by strategically minimizing the balance skewness in intermediate channels during MHPs, resulting in more balanced PCNs. To evaluate its performance, particularly in comparison to LN [36], Revive [25], and Shaduf [21], we design this experiment with a focus on the following aspects:

(1) How does Horcrux perform in the PCN concerning enhancements in the network's success ratio, volume, and alleviating channel depletion in the network?

(2) How does Horcrux perform in improving the PCN under long-term operation?

Simulation setup. Our test network is based on the snapshot of LN, the largest off-chain network, encompassing $10,529$ nodes and $38,910$ channels. Since the initial distribution of the channel balance is unknown, we assume an equal balance allocation for each channel between both participating users. This consistent approach across all experiments ensures that the balance assignment does not introduce bias into our results. We employ the payment value dataset that is randomly sampled from the Bitcoin trace from March 1 to March 31, 2021, comprising over $2.65M$ micro-payment transactions.

Evaluation methodology. We devise experimental methodologies specific to Revive [25], Shaduf [21], and Horcrux. For Revive and Shaduf, we refer to the implementations outlined in [21], including the optimized Revive (OPT-Revive), and provide clarification on Shaduf's binding strategy. Regarding Horcrux, we opt for an effective fund redistribution strategy.

In Shaduf, the rebalancing demand is resolved in a local manner, i.e., a user can transfer coins between two bound channels, cooperating with the two neighbors. To simulate this, the bindings need to be initiated for each user. The binding amount is the maximum number of coins that can be shifted between the bound channels, equally distributed to each bound channel. As for the binding strategy, we draw inspiration from the comparative experiments presented in [21]. Taking into account the costs and benefits of bindings, we opt for the 'high-to-low' binding strategy, where each user binds the channel with the highest balance to the channel with the lowest balance, the channel with the second-highest balance to the channel with the second-lowest balance, and so forth.

In Horcrux, rebalancing is also addressed locally. When a MHP occurs, each user along the path locks a certain amount of coins from its adjacent channels. Through the reallocation of these coins, efforts are made to offset the balance skewness introduced by MHPs, thereby avoiding depletion. To simulate this, we examine, at each transaction occurrence, whether the sum of balances in both channels of each intermediate user along the path meets the transaction amount. We have adopted the following strategy for fund redistribution:

We assume that for an intermediate user, the side closer to the sender is considered the left side, and conversely, the side closer to the receiver is considered the right side, with balances in the left and right channels denoted as (bal^L, bal^R). When the balance bal^L in the left channel is sufficient to cover the transaction amount α, the user transfers α from the left channel to the right channel. After the transaction is completed, the balances on the left and right remain unchanged. However, when bal^L is insufficient to cover the transaction amount α, the user transfers the entire balance bal^L from the left channel to the right channel. After the transaction is completed, the new balances in the left and right channels are $(\alpha, bal^L + bal^R)$, which results in a more balanced state compared to the previous imbalance.

Ways of evaluation and comparison. Following the methodology and with the simulation setup mentioned above, we conduct this evaluation, which includes:

(1) We test Horcrux's performance over 20 consecutive batches, with the channel capacity varying proportionally in each evaluation, to assess whether Horcrux performs better than other schemes in deposit reduction. Under the same conditions, we also tested the performance of LN, OPT-Revive, and Shaduf using the 'high-to-low' strategy (referred to as HL-Shaduf) for comparison.

(2) We test Horcrux's performance over 200 consecutive batches with a fixed channel capacity 8. For comparison, the performance of LN, OPT-Revive, and HL-Shaduf is also evaluated under the same settings.

In our simulation, we process batches of 50,000 payments each. For each payment in batches, a sender-receiver pair with the corresponding payment value is chosen at random and routed via the shortest available path. For simplicity, fees are omitted. Since randomness is introduced in the simulation, we execute each evaluation 10 times and calculate the average. As for performance metrics, we primarily utilize the success ratio of payments to assess the improvement in network throughput and the count of channels becoming depleted to evaluate the impact on network sustainability.

Evaluation result. Based on these metrics, we initialize the network model with uniformly distributed channels and statistically evaluate the performance of 4 schemes under varying conditions. The results are graphically presented as Fig. 8.

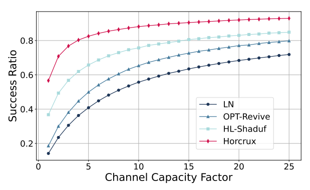
Comparison of success ratio. We vary the channel capacity from 1x to 25x and display the success ratios of 4 schemes in Fig. 8(a). Compared to LN, Revive and Shaduf show enhancements in performance ranging from 4%-8% and 13%-20% across different channel capacities, while Horcrux enhances 21%-47%. Moreover, at the lower channel capacities, Horcrux's performance can reach 2.6x, 2.0x, and 1.5x compared to LN, Revive, and Shaduf, respectively.

Comparison of depletion alleviation. We record the number of depleted channels caused by the 4 schemes under channel capacity variations from 1x to 25x, depicted in Fig. 8(b). Both Revive and Shaduf result in over a thousand depleted channels, while Horcrux's count is nearly zero. Specifically, Revive and Shaduf cause 6%-17% and 4%-13% of channel depletions in the PCN, respectively, at different capacities. The success of various MHPs enabled by Revive and Shaduf leads to imbalances in many channels. LN's notably low payment success ratio maintains balance in numerous channels through payment failures, causing its curve to overlap with Horcrux's curve in the graph. However, Horcrux shows lower numbers than LN, attributed to the benefits of fund redistribution between adjacent channels.

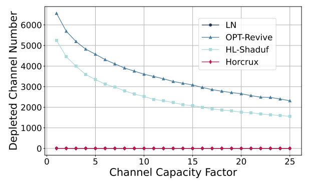
Comparison of deposit. We can observe at which scale each of the 4 schemes would achieve the same performance from Fig. 8(a). For example, at a success ratio of 70%, the channel capacity factors for LN, Revive, Shaduf, and Horcrux are 23, 13, 7, and 2, respectively. This indicates that Revive, Shaduf, and Horcrux achieve the same performance at 0.56x, 0.3x, and 0.09x capacity compared to LN. Correspondingly, they reduce the required deposit in the PCN by 43%, 70%, and 91%, respectively.

Comparison under long-term operation. The results are shown in Fig. 8(d) and Fig. 8(c). Concretely,

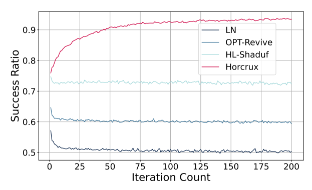
- In Fig. 8(d), as the runtime increases, LN, Revive, and Shaduf experience a decrease in performance by 1%-6% due to the PCN trending towards imbalance. In contrast, Horcrux's strategy of avoiding introducing skewness into the PCN leads to an 18% enhancement in performance, stabilizing at around



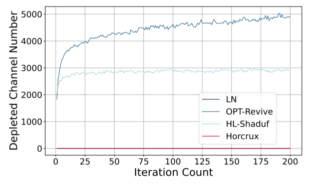
(a) Comparing the success ratio of 4 schemes under short-term operation, with running batches 20 and channel capacity factor from 1 to 25.



(b) Comparing the depleted channel number of 4 schemes under short-term operation, with running batches 20 and channel capacity factor from 1 to 25.



(c) Comparing the success ratio of 4 schemes under long-term operation, with channel capacity factor 8 and running batches from 1 to 200.



(d) Comparing the depleted channel number of 4 schemes under long-term operation, with channel capacity factor 8 and running batches from 1 to 200.

Fig. 8: Effect of Horcrux varying running times and channel capacity under uniform payment demands.

94%. Furthermore, Horcrux's performance under long-term operation increases by 43%, 33%, and 21% compared to LN, Revive, and Shaduf, respectively, which achieves 1.2x-1.5x compared to performance under short-term operation.

- In Fig. 8(c), after prolonged operation, the number of depleted channels caused by Revive and Shaduf in the PCN increased by 2% and 8%, respectively. However, the number of depleted channels caused by Horcrux after long-term operation still remains close to zero, slightly lower than LN.

In summary, compared to LN [36], Revive [25], and Shaduf [21], Horcrux exhibits an improvement in success ratio by 12%-30%, reduces user deposits by 70%-91%, and achieves an almost zero channel depletion ratio. Furthermore, Horcrux's long-term performance is 1.2x-1.5x that of the short-term performance, which indicates its continuous operation is expected to further enhance performance.

VI. Other related work

As off-chain scalability solutions, PCs and PCNs have garnered interest from both academics and industry [7, 13, 14, 17, 19, 31, 32, 36, 44, 47]. Moreover, Malavolta et al. [30] highlight that the HTLC-style solutions (e.g., [36]) suffer from wormhole attack and accordingly propose the privacy-preserving MHP construction using a new tool called Anonymous Multi-Hop Locks (AMHL). Other research on PCNs focuses on more efficient constructions [9, 20]. Unfortunately, these works require active participation from the intermediaries, which can potentially lead to unreliable, slower, costly payments and privacy concerns. PCHs, such as Tumblebit [23], A^2L [41] and BlindHub [38], rely on a single intermediary for coin exchange between the sender and receiver aiming to improve performance metrics including communication complexity and bandwidth. Furthermore, various optimization strategies for routing in PCNs have been studied [29, 37, 39, 40, 46]. The concept of VC is one of the most promising techniques to improve efficiency and reliability of PCNs by enabling the end users to establish direct off-chain channels over the intermediaries. Notable works in this area include Perun [18], LVPC [8], Elmo [26], etc. For a comprehensive understanding of the Layer-Two blockchain protocols, we refer readers to work [22].

While advancements in PCNs and MHPs have led to serious balance skewness, causing channel depletion and limiting the overall performance of PCNs [25]. The concept of rebalancing is introduced to revive depleted channels by reallocating balances between the adjacent channels. Solutions such as Revive [25], HIDE & SEEK [12], Thora [6] and Wiser [43] have made notable improvements in terms of privacy, atomicity, and efficiency, respectively. However, these cycle-based approaches suffer from significant practical limitations including path restrictions and the minimum rebalancing amount. Shaduf [21] achieves non-cycle based rebalancing by binding two adjacent channels with one on-chain transaction, allowing for the free shifting of coins between the bound channels. Unsurprisingly, these existing works primarily focus on refunding depleted channels and therefore cannot fundamentally prevent channels from being depleted.

In addition to Layer-Two optimizations, work [35] introduces a method for measuring network imbalance and a greedy heuristic algorithm to improve the local balance of each node, addressing the routing challenge posed by the opaque channel balance distributions. Merchant [45] proposes an incentive mechanism that reduces routing fees to recommend favorable transaction paths, thereby encouraging MHPs that facilitate rebalancing. Spider [40] presents a balanced multi-path routing strategy to reduce skewed payment flows, while Splicer [46] unveils a scalable routing mechanism for PCHs, enhancing throughput in both compact and expansive PCNs. Furthermore, work [28] devised a strategy for channel deposits to better accommodate in-channel payment needs.

VII. Conclusion

We present Horcrux, an efficient multi-party virtual channel protocol that tackles the fundamental issues leading to fund depletion. It offers high compatibility across various blockchains and is formally proven to be secure within the Universal Composability framework. Remarkably, Horcrux achieves enhanced reliability as it accomplishes all of this without relying on perpetual online connectivity or complex scripting support. The experimental results further confirm Horcrux's efficiency, highlighting a nearly zero depleted channel count and showcasing long-term superior performance. This establishes its cost-effectiveness and rebalancing proficiency relative to contemporary methods such as Shaduf. Additionally, Horcrux is designed to integrate smoothly with existing PCNs. Moving forward, our research will continue to focus on advancing PCN protocols that withstand channel depletion in the Byzantine adversary model.

Acknowledgment

The authors would like to thank our shepherd and the anonymous reviewers for their valuable comments. Jing was supported in part by the National Natural Science Foundation of China (No. 62172396). Peifang was supported in part by the Open Topics of Key Laboratory of Blockchain Technology and Data Security, The Ministry of Industry and Information Technology of the People's Republic of China (No. 20241112).

References

[1] Bitcoin wiki: Payment channels. Online, 2018.

[2] Real-world ln topology. https://ln.bigsun.xyz/, 2021.

[3] Implementation of horcrux. https://github.com/Anqi333/implementation-of-horcrux, 2023.

[4] 1ML. Real-time lightning network statistics. https://1ml.com/statistics, 2023.

[5] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies.* " O'Reilly Media, Inc.", 2014.

[6] Lukas Aumayr, Kasra Abbaszadeh, and Matteo Maffei. Thora: Atomic and privacy-preserving multi-channel updates. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 165–178, 2022.

[7] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part II 27*, pages 635–664. Springer, 2021.

[8] Lukas Aumayr, Matteo Maffei, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 901–918. IEEE, 2021.

[9] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure {Multi-Hop} payments without {Two-Phase} commits. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4043–4060, 2021.

[10] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Breaking and fixing virtual channels: Domino attack and donner. In *# PLACEHOLDER_PARENT_METADATA_VALUE#*, 2023.

[11] Lukas Aumayr, Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Pedro Moreno-Sanchez, and Matteo Maffei. Sleepy channels: Bi-directional payment channels without watchtowers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 179–192, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. Hide & seek: Privacy-preserving rebalancing on payment channel networks. In *International Conference on Financial Cryptography and Data Security*, pages 358–373. Springer, 2022.

[13] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Annual Cryptology Conference*, pages 421–439. Springer, 2014.

[14] Conrad Burchert, Christian Decker, and Roger Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science*, 5(8):180089, 2018.

[15] Ran Canetti. Obtaining universally compoable security: Towards the bare bones of trust. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 88–112. Springer, 2007.

[16] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Advances in Cryptology-CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings 21*, pages 19–40. Springer, 2001.

[17] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 625–656. Springer, 2019.

[18] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 106–123. IEEE, 2019.

[19] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966, 2018.

[20] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 801–815, 2019.

[21] Zhonghui Ge, Yi Zhang, Yu Long, and Dawu Gu. Shaduf: Non-cycle payment channel rebalancing. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24–28*, 2022.

[22] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised*

Selected Papers 24, pages 201–226. Springer, 2020.

[23] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017.

[24] Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Lightweight virtual payment channels. In *International Conference on Cryptology and Network Security*, pages 365–384. Springer, 2020.

[25] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 439–453, 2017.

[26] Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. Elmo: Recursive virtual payment channels for bitcoin. *Cryptology ePrint Archive*, 2021.

[27] Lightning Labs. Loop. https://lightning.engineering/loop/.

[28] Peng Li, Toshiaki Miyazaki, and Wanlei Zhou. Secure balance planning of off-blockchain payment channel networks. In *IEEE INFOCOM 2020-IEEE conference on computer communications*, pages 1728–1737. IEEE, 2020.

[29] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. *Cryptology ePrint Archive*, 2016.

[30] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. *Cryptology ePrint Archive*, 2018.

[31] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *International conference on financial cryptography and data security*, pages 508–526. Springer, 2019.

[32] Pedro Moreno-Sanchez, Arthur Blue, Duc V Le, Sarang Noether, Brandon Goodell, and Aniket Kate. Dlsag: non-interactive refund transactions for interoperable payment channels in monero. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, pages 325–345. Springer, 2020.

[33] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.

[34] Bitcoin Optech. Splicing. https://bitcoinops.org/en/topics/splicing/.

[35] Rene Pickhardt and Mariusz Nowostawski. Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2020.

[36] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.

[37] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. *White Paper*, 144, 2016.

[38] Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, O?uzhan Ersoy, Amin Sakzad, Muhammed F. Esgin, Joseph K. Liu, Jiangshan Yu, and Tsz Hon Yuen. Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2462–2480, 2023.

[39] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.

[40] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 777–796, 2020.

[41] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A 2 l: Anonymous atomic locks for scalability in payment channel hubs. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1834–1851. IEEE, 2021.

[42] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. A quantitative analysis of security, anonymity and scalability for the lightning network. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 387–396, 2020.

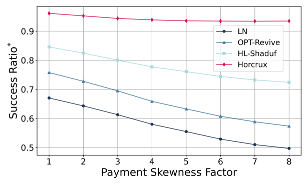
[43] Samarth Tiwari, Michelle Yeo, Zeta Avarikioti, Iosif Salem, Krzysztof Pietrzak, and Stefan Schmid. Wiser: Increasing throughput in payment channel networks with transaction aggregation. *arXiv preprint arXiv:2205.11597*, 2022.

[44] Somanath Tripathy and Susil Kumar Mohanty. Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In *International Conference on Financial Cryptography and Data Security*, pages 481–495. Springer, 2020.

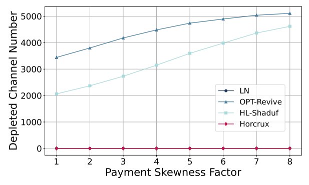
[45] Yuup Van Engelshoven and Stefanie Roos. The merchant: Avoiding payment channel depletion through incentives. In *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 59–68. IEEE, 2021.

[46] Lingxiao Yang, Xuewen Dong, Sheng Gao, Qiang Qu, Xiaodong Zhang, Wensheng Tian, and Yulong Shen. Optimal hub placement and deadlock-free routing for payment channel network scalability. 2023.

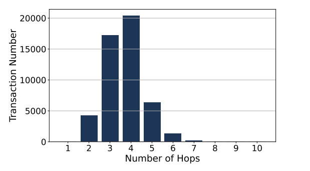
[47] Yuncong Zhang, Yu Long, Zhen Liu, Zhiqiang Liu, and Dawu Gu. Z-channel: Scalable and efficient scheme in zerocash. *Computers & Security*, 86:112–131, 2019.



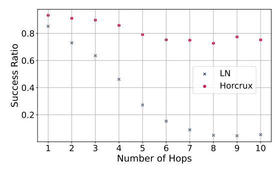
(a) Comparing the success ratio* of 4 schemes under short-term operation, with channel capacity factor 8 and payment skewness factor from 1 to 8.



(b) Comparing the depleted channel number of 4 schemes under short-term operation, with channel capacity factor 8 and payment skewness factor from 1 to 8.



(c) Transaction distribution across different number of hops in MHPs.



(d) Comparing the success ratio of LN and Horcrux under short-term operation, with channel capacity factor 8 and number of hops from 1 to 10.

Fig. 9: Evaluation of performance under skewed payments and different number of hops.

Appendix

A. Extended evaluation

Skewed payments in consumer-merchant network. Based on the test network from Section V and the LN real-world dataset, we conduct an extended performance evaluation of Horcrux in a more adversarial scenario, specifically in the consumer-merchant network, where payments are more skewed. Considering that senders only initiate transactions they can afford, we use success ratio* to measure the performance of Horcrux in comparison to LN, Revive, and Shaduf, where success ratio* is defined as the ratio of successful payments, conditioned on the senders having sufficient coins for the payments. However, the intermediate users might be poor in this payment direction, potentially leading to payment failures. The experimental results are shown in Fig. 9(a) and 9(b).

When the payment skewness factor is 8, which is extremely high in real-world scenarios, Fig. 9(a) shows that Horcrux still maintains a success ratio of over 90%, outperforming Shaduf, Revive, and LN by 1.29x, 1.63x, and 1.88x, respectively. Additionally, as illustrated in Fig. 9(b), the number of depleted channels for Horcrux remains close to zero, preserving network balance.

Impact of hop count in MHPs. We experimentally test the transaction distribution within LN across different hop counts in MHPs and evaluate the success ratio of payments for Horcrux and LN under varying hop counts. The results are shown in Fig. 9(c) and Fig. 9(d), respectively. As depicted in Fig. 9(d), Horcrux outperforms LN across all hop counts, particularly achieving a success ratio more than 14 times that of LN for hop counts exceeding 7.

B. Pseudo-code for the subroutines

In this section, we give concrete pseudo-code for the subroutines of the ideal functionality and protocol.

C. Security analysis of theorem 1

Proof. We now prove that protocol Horcrux (Fig. 7) UC-realizes the ideal functionality $\mathcal{F}$ (Fig. 6). To show the indistinguishability between the ideal world and the real world, we construct a simulator $\mathcal{S}$ to simulate the protocol Horcrux in the real world while interacting with the ideal functionality $\mathcal{F}$.

01 Freeze $[\{(pk_{i-1,i},\alpha_i^L),(pk_{i,i+1},\alpha_i^R)\},\{pk_{\mathcal{F},i}\},\{(sk_{i-1}^R, sk_i^L),(sk_i^R,sk_{i+1}^L)\}]$
It transfers the coins α_i^L of address $pk_{i-1,i}$ and coins α_i^R of address $pk_{i,i+1}$ to address $pk_{\mathcal{F},i}$ (controlled by $\mathcal{F}$) via invoking the interface Transfer$[\{(pk_{i-1,i},\alpha_i^L),(pk_{i,i+1},\alpha_i^R)\},\{(pk_{\mathcal{F},i},\alpha_i^L+\alpha_i^R)\},\{sk_{i,i-1}=sk_{i-1}^R\oplus sk_i^L, sk_{i,i+1}=sk_i^R\oplus sk_{i+1}^L\}]$ of functionality $\mathcal{F}_{\mathbb{B}}$. If the ledger $\mathcal{L}$ is updated successfully, then it responds (frz, ok).

02 Unfreeze$[\{(pk_{\mathcal{F},i},\alpha)\},\{(pk_{i-1,i},\widehat{\beta}_i^L),(pk_{i,i+1},\widehat{\beta}_i^R)\},\{sk_{\mathcal{F},i}\}]$
It transfers the frozen coins $\widehat{\beta}^L$ and $\widehat{\beta}^R$ of address $pk_{\mathcal{F},i}$ respectively to addresses $pk_{i-1,i}$ and $pk_{i,i+1}$ ($\widehat{\beta}^L+\widehat{\beta}^R=\alpha$) via invoking the interface Transfer$[\{(pk_{\mathcal{F},i},\alpha)\},\{(pk_{i-1,i},\widehat{\beta}_i^L),(pk_{i,i+1},\widehat{\beta}_i^R)\},\{sk_{\mathcal{F},i}\}]$ of functionality $\mathcal{F}_{\mathbb{B}}$. If the ledger $\mathcal{L}$ is updated successfully, then it responds $(unfrz, ok)$.

03 Update$[\widehat{\alpha},\gamma_{i-1,i},\gamma_{i,i+1}]$
- If $\widehat{\alpha}\geq 0$, it sends (ChannelUpdate,$bal_i^L := bal_i^L+\alpha_i^L, bal_i^R := bal_i^R+\alpha_i^R+fee(t)$) $\overset{r_4+2}{\hookrightarrow} U_i$ for $\widehat{\alpha}_i^L \geq \widehat{\alpha}+(n-i)fee(t)$ or (ChannelUpdate,$bal_i^L := bal_i^L+\widehat{\alpha}+(n-i)fee(t), bal_i^R := bal_i^R+\alpha_i^L+\alpha_i^R-\widehat{\alpha}-(n-i-1)fee(t)$) $\overset{r_4+2}{\hookrightarrow} U_i$ for $\widehat{\alpha}_i^L < \widehat{\alpha}+(n-i)fee(t)$.
- If $\widehat{\alpha}<0$, it sends (ChannelUpdate,$bal_i^L := bal_i^L+\alpha_i^L+fee(t), bal_i^R := bal_i^R+\alpha_i^R$) $\overset{r_4+2}{\hookrightarrow} U_i$ for $\widehat{\alpha}_i^R \geq -\widehat{\alpha}+ifee(t)$ or (ChannelUpdate,$bal_i^L := bal_i^L+\alpha_i^L+\alpha_i^R+\widehat{\alpha}-(n-i)fee(t), bal_i^R := bal_i^R-\widehat{\alpha}+ifee(t)$) $\overset{r_4+2}{\hookrightarrow} U_i$ for $\widehat{\alpha}_i^R < -\widehat{\alpha}+ifee(t)$.

04 Close $[(pk_{\mathcal{F},i},sk_{\mathcal{F},i}),\widehat{\alpha},\gamma_{i-1,i},\gamma_{i,i+1}]$
- If $\widehat{\alpha}\geq 0$, it invokes subroutine Unfreeze $[\{(pk_{\mathcal{F},i},\alpha)\},\{(pk_{i-1,i},\alpha_i^L-\widehat{\alpha}-(n-i)fee(t)),(pk_{i,i+1},\alpha_i^R+\widehat{\alpha}+(n-i)fee(t))\},\{sk_{\mathcal{F},i}\}]$ for $\widehat{\alpha}_i^L\geq\widehat{\alpha}+(n-i)fee(t)$ or Unfreeze $[\{(pk_{\mathcal{F},i},\alpha)\},\{(pk_{i-1,i},0),(pk_{i,i+1},\alpha)\},\{sk_{\mathcal{F},i}\}]$ for $\widehat{\alpha}_i^L < \widehat{\alpha}+(n-i)fee(t)$ and sends (Closed,$\widehat{\gamma}_{0,n}$) $\overset{r_4+4}{\hookrightarrow} U_i$.
- If $\widehat{\alpha}<0$, it invokes subroutine Unfreeze $[\{(pk_{\mathcal{F},i},\alpha)\},\{(pk_{i-1,i},\alpha_i^L-\widehat{\alpha}+ifee(t)),(pk_{i,i+1},\alpha_i^R+\widehat{\alpha}-ifee(t))\},\{sk_{\mathcal{F},i}\}]$ for $\widehat{\alpha}_i^R\geq-\widehat{\alpha}+ifee(t)$ or Unfreeze $[\{(pk_{\mathcal{F},i}\alpha)\},\{(pk_{i-1,i},\alpha),(pk_{i,i+1},0)\},\{sk_{\mathcal{F},i}\}]$ for $\widehat{\alpha}_i^R < -\widehat{\alpha}+ifee(t)$ and sends (Closed,$\widehat{\gamma}_{0,n}$) $\overset{r_4+4}{\hookrightarrow} U_i$.

Fig. 10: The subroutines of ideal functionality $\mathcal{F}$

At the beginning, $\mathcal{S}$ corrupts some users of $\{U_i\ (i\in[0,n])\}$ as $\mathcal{A}$ does (notice that only one of users U_0 and U_n can be

Assumption: $\oplus := +$ if $SIG \in \{Schnorr, BLS\}$ and $\oplus := *$ if $SIG = ECDSA$

01 OpenAccount(P_0, P_1, P_2)

It takes privet inputs $\widehat{sk}_0^R$, $\widehat{sk}_1$ and $\widehat{sk}_2^L$ held by users P_0, P_1 and P_2 respectively.

- It sets secret key as $\widehat{sk}_{0,1,2} := \widehat{sk}_0^R \oplus \widehat{sk}_1 \oplus \widehat{sk}_2^L$;
- It generates a three-party account as $\widehat{pk}_{0,1,2} := g^{\widehat{sk}_{0,1,2}}$.

....

02 CheckValid(tx^{fud})

It sets funding transaction as $tx_k^{fud}[\{(pk_{k-1,k}, \alpha_k^L), (pk_{k,k+1}, \alpha_k^R)\}, \{(\widehat{pk}_{k-1,k,k+1}, \alpha_k^L + \alpha_k^R)\}]$, where $k \in [0, n]$.

- It checks that $bal_k^L \geq \alpha_k^L$ and $bal_k^R \geq \alpha_k^R$;
- It checks that $\alpha_k^L + \alpha_k^R \geq \alpha + Max\{i, n-i\} \cdot fee(T)$.

....

03 UpdateChannel$(\gamma_{k,k+1}, \beta_{k,k+1}, \beta_k^R, \beta_{k+1}^L)$

- It sets the value of PC $\gamma_{k,k+1}$ between users U_k and U_{k+1} as $\beta_{k,k+1}$, and the balance of U_k is $bal_k^R := \beta_k^R$ and the balance of U_{k+1} is $bal_{k+1}^L := \beta_{k+1}^L$;
- It updates PC state as $st_{k,k+1} := tx[\{pk_{k,k+1}, bal_k^R + bal_{k+1}^L\}, \{(pk_k, bal_k^R), (pk_{k+1}, bal_{k+1}^L)\}]$, where accounts pk_k and pk_{k+1} are owned by users U_k and U_{k+1} respectively;
- It invokes MultiSign$(\{sk_k, sk_{k+1}\}, st_{k,k+1})$ and obtains signature $\sigma_{k,k+1}$, where $(sk_k \oplus sk_{k+1}, pk_{k,k+1}) \in SIG.KGen(\lambda)$.

....

04 MultiSign$(\{sk_1, \cdots, sk_\ell\}, m)$

It takes privet inputs $sk_1, \cdots, sk_\ell$ held by users $P_1, \cdots, P_\ell$ respectively.

- It sets secret key as $sk_{1,\cdots,\ell} := sk_1 \oplus \cdots \oplus sk_\ell$ and public key as $pk_{1,\cdots,\ell} := g^{sk_{1,\cdots,\ell}}$;
- It computes signature $SIG.Sign(sk_{1,\cdots,\ell}, m) \rightarrow \sigma_{1,\cdots,\ell}$ and sends it to users $P_1, \cdots, P_\ell$;
- Users $P_1, \cdots, P_\ell$ respectively verify if $SIG.Vrf(pk_{1,\cdots,\ell}, m, \sigma_{1,\cdots,\ell}) \rightarrow 1$, and abort otherwise.

....

05 Redistribute$(\widehat{\alpha}_\mu, st_{vc})$

- If $\widehat{\alpha}_\mu \geq 0$, it sets $\alpha_i^{'L} := \alpha_i^L - \widehat{\alpha}_i^R - fee(t_\mu)$, and $b_i^R = 1$ if $\alpha_i^{'L} \leq 0$ and otherwise $b_i^R = 0$.
  - It with U_{i+1} updates channel $\gamma_{i,i+1}$'s state as `UpdateChannel`$(\gamma_{i,i+1}, bal_i^{'R} + bal_{i+1}^{'L} + \alpha_i^L + \alpha_i^R - b_i^R\alpha_i^{'L} + b_{i+1}^R\alpha_{i+1}^{'L}, bal_i^{'R} + \alpha_i^L + \alpha_i^R - b_i^R\alpha_i^{'L}, bal_{i+1}^{'L} + b_{i+1}^R\alpha_{i+1}^{'L})$, generates close transaction $tx_i^c[\{\widehat{pk}_{i-1,i,i+1}, \alpha_i^L + \alpha_i^R\}, \{(pk_{i-1,i}, b_i^R\alpha_i^{'L}), (pk_{i,i+1}, \alpha_i^R + \alpha_i^L - b_i^R\alpha_i^{'L})\}]$ and sends tx_i^c to U_{i-1} and U_{i+1}.
- If $\widehat{\alpha}_\mu < 0$, it sets $\alpha_i^{'R} := \alpha_i^R - \widehat{\alpha}_i^L - fee(t_\mu)$, and $b_i^L = 1$ if $\alpha_i^{'R} \leq 0$ and otherwise $b_i^L = 0$.
  - It with U_{i-1} updates channel $\gamma_{i-1,i}$'s state as `UpdateChannel`$(\gamma_{i-1,i}, bal_{i-1}^{'R} + bal_i^{'L} + \alpha_i^L + \alpha_i^R - b_i^L\alpha_i^{'R} + b_{i-1}^L\alpha_{i-1}^{'R}, bal_{i-1}^{'R} + b_{i-1}^L\alpha_{i-1}^{'R}, bal_i^{'L} + \alpha_i^L + \alpha_i^R - b_i^L\alpha_i^{'R})$, generates close transaction $tx_i^c[\{\widehat{pk}_{i-1,i,i+1}, \alpha_i^L + \alpha_i^R\}, \{(pk_{i-1,i}, \alpha_i^L + \alpha_i^R - b_i^L\alpha_i^{'R}), (pk_{i,i+1}, b_i^L\alpha_i^{'R})\}]$ and sends tx_i^c to U_{i-1} and U_{i+1}.

Fig. 11: The subroutines of the protocol

corrupted). We begin with the real-world protocol execution, gradually change the simulation in these hybrids and then argue about the proximity of neighboring experiments.

Hybrid $\mathcal{H}_0$: It is the same as the real world protocol execution (Fig. 7).

Hybrid $\mathcal{H}_1$: It is the same as the above execution except that the protocol `MultiSign` in procedures (A) and (C) of generating signatures jointly by 2 or 3 users is simulated using the MPC simulator $\mathcal{S}_{MPC}$ for the corrupted users (notice that such a simulator exists for a secure MPC protocol `MultiSign`).

Hybrid $\mathcal{H}_2$: It is the same as the above execution except that the adversary corrupts user U_i and obtains a new state of the underlying PC before the simulator cooperates the update operation on behalf of user U_{i-1} or U_i, the simulator aborts.

Hybrid $\mathcal{H}_3$: It is the same as the above execution except that the adversary corrupts user U_i and posts a valid funding transaction before the simulator cooperates the multi-sign operation on behalf of users U_{i-1} and U_{i+1}, the simulator aborts.

Hybrid $\mathcal{H}_4$: It is the same as the above execution except that the adversary corrupts user U_i $(i \in \{0, n\})$ and generates a valid VC state before the simulator initiates its signing operation on behalf of user U_{n-i}, the simulator aborts.

Hybrid $\mathcal{H}_5$: It is the same as the above execution except that the adversary corrupts user U_i and obtains a valid closing transaction tx_i^c before the simulator cooperates the multi-sign operation on behalf of users U_{i-1} and U_{i+1}, the simulator aborts.

Hybrid $\mathcal{H}_6$: It is the same as the above execution except that the adversary corrupts user U_i and outputs a valid closing transaction tx_i^c. The simulator outputs tx_{i-1}^c or tx_{i+1}^c, where $\Sigma_{SIG}.Vf(pk_{i-2,i-1,i}, tx_{i-1}^c, \sigma_{i-2,i-1,i}) \neq 1$ or $\Sigma_{SIG}.Vf(pk_{i,i+1,i+2}, tx_{i+1}^c, \sigma_{i,i+1,i+2}) \neq 1$, the simulator aborts.

Simulator $\mathcal{S}$. The simulator $\mathcal{S}$ is defined as the execution in $\mathcal{H}_6$ while interacting with the ideal functionality $\mathcal{F}$.

Below, we show the indistinguishability between $\mathcal{H}_0$ and $\mathcal{H}_6$. In addition, we use $\approx_c$ to denote computational indistinguishability for a PPT algorithm.

$\mathcal{H}_0 \approx_c \mathcal{H}_1$. The indistinguishability directly follows from the security of protocol `MultiSign`. The security of protocol `MultiSign` for multi-party signature generation guarantees the existence of $\mathcal{S}_{MPC}$.

$\mathcal{H}_1 \approx_c \mathcal{H}_2$. The only difference between the two hybrids is that in $\mathcal{H}_2$ the simulator aborts, if the corrupted user U_i can generate a valid multi-party signature of a PC state itself.

$\mathcal{H}_2 \approx_c \mathcal{H}_3$. The only difference between the two hybrids is that in $\mathcal{H}_3$ the simulator aborts, if the corrupted user U_i can generate a valid multi-party signature of a funding transaction itself.

$\mathcal{H}_3 \approx_c \mathcal{H}_4$. The only difference between the two hybrids is that in $\mathcal{H}_4$ the simulator aborts, if the corrupted user U_i $(i \in [0, n])$ can generate a valid multi-party signature of a VC state itself.

$\mathcal{H}_4 \approx_c \mathcal{H}_5$. The only difference between the two hybrids is that in $\mathcal{H}_5$ the simulator aborts, if the corrupted user U_i can generate a valid multi-party signature of closing transaction itself.

$\mathcal{H}_5 \approx_c \mathcal{H}_6$. The only difference between the two hybrids is that in $\mathcal{H}_6$ the simulator aborts, if the adversary posts a valid closing transaction, the simulator cannot post its valid closing transaction. With the security the underlying blockchain and PC, and the rational adversarial model, the probability of the event triggered in $\mathcal{H}_6$ is negligible, otherwise, the malicious U_i will lose coins for the closing transaction tx^c_{i-1} or tx^c_{i+1} is not finally confirmed on chain, which results in that the its PC $\gamma_{i-1,i}$ or $\gamma_{i,i+1}$ cannot be updated correctly.