Shaduf++: Non-Cycle and Privacy-Preserving Payment Channel Rebalancing*

Zhonghui Ge, Yi Zhang, Yu Long and Dawu Gu
Shanghai Jiao Tong University
{zhonghui.ge, zy0105, longyu, dwgu}@sjtu.edu.cn

***Abstract*—A leading approach to enhancing the performance and scalability of permissionless blockchains is to use the payment channel, which allows two users to perform off-chain payments with almost unlimited frequency. By linking payment channels together to form a payment channel network, users connected by a path of channels can perform off-chain payments rapidly. However, payment channels risk encountering fund depletion, which threatens the availability of both the payment channel and network. The most recent method needs a cycle-based channel rebalancing procedure, which requires a fair leader and users with rebalancing demands forming directed cycles in the network. Therefore, its large-scale applications are restricted.**

In this work, we introduce Shaduf, a novel non-cycle off-chain rebalancing protocol that offers a new solution for users to shift coins between channels directly without relying on the cycle setting. Shaduf can be applied to more general rebalancing scenarios. We provide the details of Shaduf and formally prove its security under the Universal Composability framework. Our prototype demonstrates its feasibility and the experimental evaluation shows that Shaduf enhances the Lighting Network performance in payment success ratio and volume. Experimental results also show that our protocol prominently reduces users' deposits in channels while maintaining the same amount of payments. Moreover, as a privacy enhancement of Shaduf, we propose Shaduf++. Shaduf++ not only retains all the advantages of Shaduf, but also preserves privacy for the rebalancing operations.

I. Introduction

Blockchain has seen rapid development over the last decade, enabling secure transactions in a distributed and trustless setting. Complex transaction logic is also supported with the help of so-called *smart contracts*, which can be regarded as program codes executed on the blockchain. However, the scalability issue still limits the deployment of blockchain, due to the inherent consensus mechanism to achieve a consistent view on transactions among all peers. For example, Bitcoin processes tens of transactions per second and requires around one hour to confirm a transaction.

A novel solution, called *payment channels*, has been introduced to address the scalability challenge. A payment channel allows two users to deposit funds to a co-maintained ledger and perform payments without broadcasting and recording them on the blockchain. It is guaranteed that users could refund coins in the two-party ledger to the blockchain at any time. Since payments require only approval from channel users rather than all peers of the blockchain, they are conducted quickly and confirmed immediately, only limited by the communication bandwidth and latency.

Payment channels are extended to the network, in which users sharing no channels can perform multi-hop payments leveraging paths of existing channels. With other users as relays of payments, users do not have to open channels to each other, thereby reducing costs in maintaining channels. Lightning Network (LN) [23], for example, is a practical payment channel network deployed upon Bitcoin.

Users in payment channels face the problem of fund depletion [14]. As payment flows in both directions over a channel are not equal, funds gradually accumulate in the direction of the higher flow. Eventually, the channel becomes depleted and fails payments in that direction, undermining the off-chain network throughput. Current solutions for refunding the depleted channel fall into two categories. The first one is to refund the single channel using on-chain coins. A trivial approach is to close and reopen the channel, which requires two costly and time-consuming on-chain transactions. LOOP (see https://lightning.engineering/loop/) reduces the refund costs to one on-chain transaction. Both approaches need interactions with the blockchain each time of channel refunding, which can be regarded as "one-time on-chain execution, one-time refunding". The second solution is to refund the channel by reallocating deposits in adjacent channels, also called *"rebalance"*. It recovers the channel from depletion without performing transactions on the blockchain, exemplified by Revive [14]. Relying on circular paths in the payment channel network, Revive instructs coins to flow along the path, therefore users' coins are rebalanced from well-funded channels to poorly-funded channels. The whole process is executed off-chain, thus being cost-free for unlimited times of rebalancing and relieving the transaction load on the underlying blockchain. These features make Revive act well in scenarios where users have stable and circular rebalancing relations.

Nevertheless, Revive suffers from low feasibility in large-scale applications, such as LN, because of the lack of *autonomy*. Specifically, to achieve a successful off-chain rebalancing, the following requirements need to be met. (1) For users with the demand of rebalancing, both the channels they want to rebalance and the direction of desired coin flows can form directed cycles in the payment channel network. (2) A fair leader is necessary to collect users' rebalancing demands, find the directed cycles, and generate the rebalancing transactions for cycles justly. (3) Once a rebalancing cycle is established, all cycle users would cooperate with the rebalancing procedure. If any of these requirements are not satisfied, Revive will fail. To make matters worse, even if the rebalancing succeeds, only the

*Part of the work appeared at the 29th Network and Distributed System Security Symposium - NDSS 2022.

TABLE I: Comparison of channel refunding solutions.

	# On-Chain TXs for n Times Refunding	Wait Time for 1 Time Refunding	Leader	Bitcoin-compatible	Usability		
					Non-cycle Users		Cycle Users
					Single-channel Users	Multiple-channel Users	
Close-and-Reopen	$2n$	2Δ	No need	Yes	●	●	●
LOOP	n	Δ	No need	Yes	●	●	●
Revive	**0**	**instantly**[1]	Fair[2]	Yes[3]	○	○	◐
Shaduf	**1**[4]	**instantly**[1]	No need	No	○[5]	●	●

Δ denotes the blockchain delay, which is the maximum time required for one transaction to be recorded on the blockchain.
○: unusable; ◐: usable when requirements are met; ●: usable.

[1] Instantly when involved users cooperate. In Revive, it includes users along the cycle. In Shaduf, it includes two adjacent users.

[2] The leader treats users' rebalancing requests equally.

[3] With the atomic multi-channel update protocol introduced in [10], Revive could be deployed to Bitcoin-compatible platforms.

[4] For one single pair of channels, Shaduf achieves 1-time on-chain binding, n-times off-chain refunding.

[5] Although the single-channel users cannot be off-chain rebalanced, their channels could help other users to rebalance. In the LN snapshot used in the evaluation (Section VI-B), the single-channel users account for around 45%.

minimum rebalancing amount among the cycle users could be achieved. In other words, one user's rebalanced amounts are limited by the amounts of other users in the cycle.

Contributions and roadmap. This work addresses the aforementioned shortcomings with a protocol we call *Shaduf*, which overcomes the limitations of Revive and can be applied to more general rebalancing scenarios. Our contributions include,

- *Novel solution for payment channel rebalancing.* We introduce Shaduf, a non-cycle off-chain channel rebalancing scheme, which allows users to perform unlimited times of off-chain coin shifts between channels after *one-time on-chain binding*. Compared with Revive, Shaduf does not rely on cycles in the network. Even better, since only the user's own channels and adjacent users are involved, Shaduf is resistant to uncooperative participants and not restricted by others' demands. The comparison between current approaches and Shaduf is shown in Table I. An additional benefit brought by the autonomy of rebalancing is that Shaduf reduces the deposits of channel users and maintains the payment amount (Section III and V).
- *Formal security proof.* We formalize the security properties of Shaduf using the ideal/real world paradigm in the Universal Composability (UC) framework and provide the formal security proof according to our definition (Section IV and V, and Appendix C).
- *Implementation and valid evaluation.* We provide a proof-of-concept implementation of Shaduf on Ethereum to show the required one-time binding is cheap compared with the unlimited times of rebalancing brought by it. Besides, an experimental evaluation demonstrates that Shaduf outperforms LN and Revive. Particularly, under different application strategies, Shaduf increases by 6% - 28% in terms of payment success ratio and volume, and reduces users' deposits by 37% - 75% while maintaining the same network performance (Section VI).
- *Privacy enhancement.* Furthermore, we discuss the privacy leakage problems in Shaduf and introduce Shaduf++, which achieves not only the same rebalancing effects as Shaduf, but also privacy protection as well (Section VII).

II. Preliminaries

A. Blockchain and Smart Contracts

The blockchain is an append-only transaction ledger and maintains users' balances. Let Δ be the blockchain delay, i.e., the time for a transaction to be recorded on the blockchain.

A smart contract is a piece of code executed on the blockchain, receiving coins and handling them according to predefined rules when triggered by users or other contracts. Our scheme is constructed upon Turing-complete blockchains such as Ethereum.

B. Payment Channel

There are three phases in the lifetime of a payment channel: open, update, and close, as shown in Figure 1.

1) Open: Two users, Alice and Bob, open a channel by depositing funds into it. Firstly, Alice sends a coins to the contract as her deposits in the channel. Upon receiving the coins, the contract informs Bob of the opening event and waits Δ time for his confirmation. If Bob confirms, the channel is opened and the sent along b coins serve as his deposits in the channel. Otherwise, Alice retrieves the coins she deposited.

2) Update: Two users update the channel state in pace with off-chain payments. The channel state includes the allocation of the channel capacity, i.e., a total of $a+b$ coins, between Alice and Bob, and a version that indicates the number of updates and increases by 1 each update. A state is valid when it is confirmed by both users, that is, users have exchanged their signatures on the state.

Any user can initiate the update. For instance, when Alice performs a payment, she increases the channel version, updates

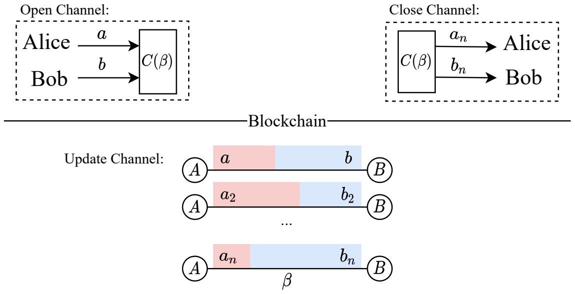
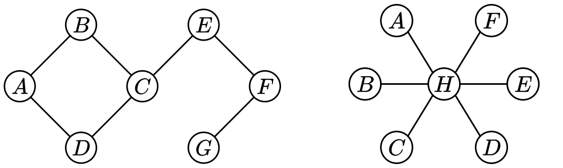


Fig. 1: Three phases of a payment channel, exemplified by β whose users are Alice and Bob. Dashed boxes indicate on-chain transactions and colored boxes represent the balances of users in the channel. Two users first escrow a and b coins to the contract $\mathcal{C}(\beta)$ respectively, thereby opening the channel. After an arbitrary number of payments, they close the channel, and the latest balances, a_n and b_n, are refunded.



(a) Payment via a path. (b) Payment via a hub.

Fig. 2: Two forms of multi-hop payments in the payment channel network: payment via a path and via a hub. Lines between users denote channels.

the balance allocation, and sends the updated state to Bob along with her signature on it. If Bob approves the state and replies with his signature on it, the state becomes valid. The process is completed without interacting with the blockchain.

3) Close: Users could close the channel and withdraw their balances to the blockchain. In particular, it happens when users enter into a dispute, such as when Alice receives no response after sending the updated state to Bob. The user who wants to close the channel, suppose Alice, submits the request to contract along with the latest valid channel state in her view. The contract informs Bob of the closing and waits Δ for his latest channel state. Comparing received valid states, the contract decides on the more recent one, i.e., the state with a higher version, and dispenses funds accordingly.

C. Payment Channel Network

The payment channel network (PCN) is constituted by payment channels, where users could perform payments taking others as intermediaries. Figure 2a presents a typical payment channel network structure, where channels are established between users who perform payments frequently. For example, C opens channels with B, D, and E. The users who share no channels, such as A and E, can perform payments via a multi-hop path connecting them. In this figure, A could pay E via paths A-B-C-E and A-D-C-E. Here we ignore mechanisms guaranteeing the security of the multi-hop payments and only consider the path. When all users along the path have sufficient balances, the payment can be completed.

However, it is non-trivial to find a fund-sufficient path between users, because the balances of users in a channel are invisible to others. Payment hub relieves the burden in finding paths, as shown in Figure 2b. Users setup channels with the hub, which in turn links any pair of users. Although the path is easy to be settled, H needs to escrow a huge deposit to channels to afford payments between these users, impeding its deployment in large-scale networks.

D. Rebalancing in PCN

Payment channels face the challenge of fund depletion, where funds in the channel (i.e., channel capacity) flow to one user. It reduces a channel from bi-directional to unidirectional. Refunding the depleted channel using on-chain coins needs interactions with the blockchain, which is costly and requires at least Δ time wait before the channel is refunded. Considering the case where one user serves in multiple channels, Revive [14] proposes to refund a channel leveraging coins in adjacent channels without interactions with the blockchain.

We illustrate how Revive works by taking the directed cycle $A{\rightarrow}B{\rightarrow}C{\rightarrow}D{\rightarrow}A$ in Figure 2a as an example. Assume each user has adequate coins in the channel with the next user and lacks funds in the channel with the previous user. Therefore, each user has the demand to rebalance coins between its two channels. Revive performs the coin flow along the directed cycle, where each user sends coins to the next user and receives coins from the previous one. To enable the cooperation of cycle participants, only the minimum rebalancing demand of them can be satisfied. After that, for each user, there are balanced coins in its adjacent channels and the sum of coins in the two channels remains unchanged.

Compared with Close-and-Reopen and LOOP, while Revive is attractive for the cost-free feature and unlimited times of rebalancing, it suffers from the lack of autonomy. Firstly, it is only possible when users with the rebalancing demands could form directed cycles in the network. Users, E, F, G in Figure 2a and all users in Figure 2b, can neither take the solution nor facilitate others' rebalancing. Users, A, B, C, D in Figure 2a could be rebalanced only when all of them have the rebalancing demands in the consistent direction. Secondly, the achievement and amount of rebalancing are both restricted by the cycle participants, including the non-adjacent ones. Therefore, Revive's application in PCN is limited.

E. Cryptography Tools

1) Commitments: A commitment scheme allows a prover to commit a message without revealing it, and open the commitment with the message later. It consists of two algorithms, Commit and Open. In the phase of committing, the prover runs the Commit algorithm on the message m such that $\mathsf{cm} \leftarrow \mathsf{Commit}(m; r)$, where r is the randomness. Then the prover sends the commitment cm to the verifier. In the phase of opening, the prover sends the message m and the randomness r to the verifier, which verifies that whether m is the committed value by running the Open algorithm such that $1/0 \leftarrow \mathsf{Open}(\mathsf{cm}, m, r)$.

The commitment scheme enables that one commitment cannot be opened to two values, and the verifier could not derive the committed value from the commitment. We use the Pedersen commitment scheme [22] in our protocol, which also has the *homomorphic* property, i.e., $\mathsf{Commit}(m_1; r_1) + \mathsf{Commit}(m_2; r_2) = \mathsf{Commit}(m_1 + m_2; r_1 + r_2)$. We refer readers to [22] for more details.

2) Range Proof: A range proof allows the prover to convince the verifier that the committed value lies in a certain interval without leaking any information about the value, except the fact that it lies in the interval. The language is defined as $\mathcal{L} = \{\mathsf{cm} | \exists m, r, s.t. (\mathsf{cm}) = \mathsf{Commit}(m; r) \wedge m \in [a, b]\}$. It consists of two algorithms, $\pi \leftarrow \mathcal{P}(x, w)$ which outputs the proof π on input the statement x and its witness w, and $1/0 \leftarrow \mathcal{V}(x, \pi)$, which outputs accept or reject on input the statement x and the proof π.

The range proof scheme has three properties, *completeness*, *soundness*, and *zero-knowledge*. Intuitively, the completeness states that an honest prover could always generate a proof that is accepted by the verifier. The soundness implies that the prover could not generate an accepted proof for an incorrect statement. The zero-knowledge states that the verifier could not know any additional information except that the statement is correct. We use the Bulletproof scheme in our work and refer readers to [4] for further details.

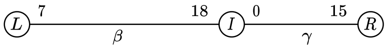
III. Shaduf Overview

While current solutions in rebalancing channels rely on a set of participants who can form a cycle in the network, Shaduf allows one user to shift coins between adjacent channels in a non-cycle setting. In a nutshell, after one-time on-chain binding, a user can shift coins between two *bound channels* directly, for unlimited times and in both directions.

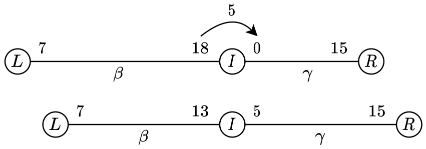
In this section, we first demonstrate the basic construction and introduce the one-time channel binding. Then we describe the rebalancing procedures and extend the structure in the payment channel network. The formal and detailed description of Shaduf is presented in Section IV and V.

A. Basic Construction Idea

Consider the basic structure where one user opens two channels with different users, as shown below.



User I setup channel β with L, and channel γ with R. Currently, I owns sufficient coins, say 18 coins, in β and 0 coins in γ. Since the money has been run out, I cannot initiate payments in γ. In Shaduf, I refunds it leveraging coins in β, depicted in the below example.



I shifts 5 coins from β to γ, with which I could perform further payments in γ. The coin shift is completed off-chain between the two channels directly, without the involvement of the blockchain or other participants. Consequently, the rebalancing is executed at no cost and of high efficiency. Moreover, it can be extended to multiple times of coin shift in both directions. For example, I can shift coins from β to γ and shift them back later then.

During the process, measures should be taken to guarantee the balance security, i.e., honest users would not lose money [8], [17]. For instance, to guarantee that there are sufficient coins to be shifted out of one channel and these coins are accepted in another channel for further payments, each coin shift needs approval from the other two users. In addition, to guarantee the off-chain rebalancing takes effect, the off-chain shifted coins should be on-chain delivered at the time of claim.

B. Necessity of One-Time Channel Binding

Simple as it seems, the basic scheme suffers from the "double-shifting attack", where I colludes the other channel user and shifts coins to multiple channels. For example, I shifts 5 coins from β to γ and 21 coins to another channel η, which is composed of I and P. A total of 26 coins are shifted out of β, while there are only 25 coins in it, leading to 1 coin loss at the time of claim in either γ or η.

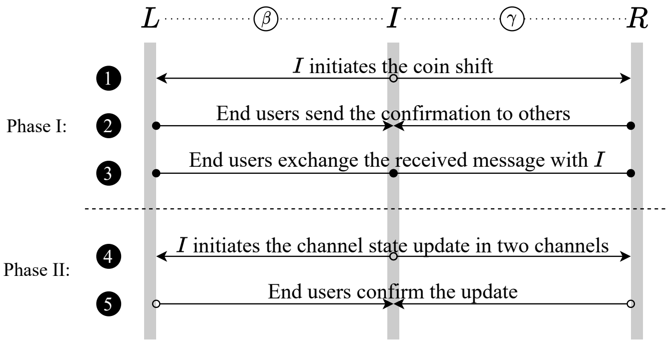

L
β
I
γ
R
Phase I:
1
I initiates the coin shift
2
End users send the confirmation to others
3
End users exchange the received message with I
Phase II:
4
I initiates the channel state update in two channels
5
End users confirm the update

Fig. 3: Flowchart of rebalancing between bound channels β and γ, which are composed by users $\{L, I\}$ and $\{I, R\}$, respectively. The first phase is bind state update, initiated by I. The second phase is channel state update, initiated by I in two channels separately. For a solid line, the arrow indicates the recipient of the message, and the hollow circle indicates the sender. For users who both send and receive messages at a step, they are represented by solid circles.

Different from other off-chain operations that only concern the balance allocation within one channel, the coin shifts in our scheme change the funds reserved in the two channels. More concretely, fund distribution between the two channels. To solve this problem, it needs to be declared on the blockchain before performing off-chain coin shifts that funds in β and γ can be redistributed along with I's coin shifts between them. The process is called *"bind"*, whose introduction maintains the low cost of basic construction and guarantees the balance security. Two channels are bound when the bind is recorded on the blockchain.

After the one-time bind, multiple times of off-chain coin shifts in both directions are allowed. If any user wants to claim the shifted coins, he could request the blockchain to *"unbind"* the two channels. We refer to Section V for the detailed description of the bind and unbind processes.

C. Rebalancing Procedures after Binding

After the channel binding, unlimited times of rebalancing between two bound channels are enabled. In this way, a record of coin shifts, called bind state, needs to be maintained by the three users. Each time performing the coin shift, users need to update the bind state and channel state, as shown in Figure 3.

Before describing the two phases, we first clarify the contents of the bind state and channel state. Besides the resulting coin shift between channels, two parts are included in a bind state: the bind version which is increased by 1 each shift to indicate the number of shifts, and the latest coin shift for the security consideration which is explained later. In the channel state, besides the channel version and balances of users, the bind version is also included to indicate the latest bind state, since the shifted coins are also part of the channel funds.

In the first phase, three users achieve agreement on the updated bind state, which is initiated by I. Assume that I wants to shift c coins from β to γ. I updates the bind state, including recalculating the total coin shift, recording this shift as the latest one, and increasing the bind version by 1. Then I sends the new state to L and R for their approval (Step (1)).

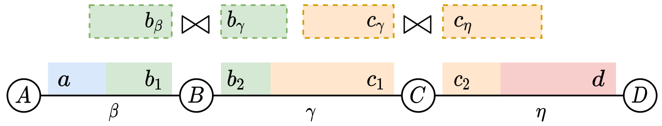


Fig. 4: Extension in payment channel network. Colored boxes represent coins of users and the length corresponds to the value. The dashed boxes upon a channel denote the bound amounts and $\bowtie$ connects bound channels. B binds b_β coins in β to b_γ coins in γ, both values are higher than B's balances in the channels, denoted as b_1 and b_2, respectively. C binds c_γ coins in γ to c_η coins in η, the former value is lower than C's balance c_1 in γ and the latter is higher than c_2 in η.

From the view of L, it needs to be verified that I has more than c coins, while R needs to ensure that there are no less than c coins in β currently. Since the channel capacity is public and the previous coins shifts are known, R could do the verification locally. It is symmetrical when coins are shifted in the opposite direction. When verification is passed, L and R reply to I with signatures on the updated bind state. In addition, they forward it to each other to inform that they acknowledged the update (Step (2)). To avoid that L or R send different messages to I and the other one, leading to an inconsistent view on whether the update is confirmed, I exchanges the received message with each other (Step (3)). Users proceed to the second phase when the new bind state is approved by all of them.

In the second phase, users update the state of each channel. Especially, the bind version and I's balance recorded in the channel state need to be recalculated. I initiates the channel state update in the two channels separately (Step (4)). In the updated channel state, I's balance is increased by c in γ while it is decreased by c in β, the channel version and the bind version are all increased by 1. Receiving the correctly updated channel state, the other user replies with the approval and completes the rebalancing (Step (5)).

Now we discuss cases where the process halts and what an honest user, denoted as P, should do. When it halts in the first phase, if P has not sent the signature on the updated bind state to others, the halt has no influence since the state is invalid yet. If P has sent the signature but not received the other's signature, then it is not sure whether the coin shift takes effect. At this point, P initiates a dispute on the blockchain, which informs the other users to submit the latest bind state in their view and decides the one with the highest version. From the state, P knows whether the coin shift is effective. When it halts in the second phase, it means that the bind state has been updated but the channel state update is refused by the other one in the channel. At that time, P sends the request of state update to the blockchain along with the latest bind state (if it has not been submitted) and channel state. The blockchain would verify that the bind version included in the bind state is 1 greater than that included in the channel state, indicating I's balance has not been updated. Then the blockchain updates it according to the last coin shift recorded in the bind state, completing the rebalancing.

D. Extension in PCN

The basic structure introduced above has limited application in the payment channel network. On the one hand, among the three users of the two bound channels, only the intermediate can shift coins in the two channels, while the other two users cannot send or receive their coins to other channels. On the other hand, the intermediate can only shift coins between the two channels and cannot interact with a third channel. The above limitations are caused by the fact that one channel can only be bound to another channel. To relieve these limitations, one channel needs to be allowed to bind multiple channels, taking either end user as the intermediate.

Recall in the basic structure, it is required that the upper limit of coins that can be shifted out of a channel is its capacity. If the upper limit is set to be lower than the capacity, the remaining amount of coins can be shifted to other channels. In this way, one channel can be bound to multiple channels. When two channels are bound, the upper limit of funds that could be shifted to each other should be specified. In addition, when one channel is bound to multiple channels, the sum of coin shift limits to each channel should be no greater than the channel capacity to guarantee the coins can eventually be delivered. In the following, we demonstrate the extension in two kinds of payment in the off-chain network, payments via a path and a hub.

1) Payment via a path: Users without direct channels can conduct off-chain payments leveraging other users as intermediates. Figure 4 presents an example where A and D conduct payments through B and C. Currently, B has b_1 coins in β and b_2 coins in γ, and C has c_1 coins in γ and c_2 coins in η. Assuming that A has sufficient coins in β, the coins that can be paid by A to D is $min(b_2, c_2)$.

The bind between the three channels are as follows: B binds b_β coins of β to b_γ coins of γ, C binds c_γ coins of γ to c_η coins in η. We first analyze the bind between channels, taking γ as an example. γ is bound to β taking B as the intermediate and b_γ as the upper limit of coins that can be shifted out. Although the upper limit is b_γ, B has only b_2 coins, then B can shift at most $min(b_2, b_\gamma) = b_2$ coins out of γ. Here we assume there are no previous shifts. If there are, such as a total of e_β coins from β to γ, or e_γ coins from γ to β, the coins that can be shifted are $min(b_2, b_\gamma + e_\beta)$ or $min(b_2, b_\gamma - e_\gamma)$. It holds the same for coin shift from γ to η. Besides, due to the fact that the sum of the upper limits $b_\gamma + c_\gamma$ is less than the capacity $b_2 + c_1$, the remaining quota can be utilized in binding other channels.

Since B can shift at most b_1 coins from β to γ and C can shift at most $min(c_1, c_\gamma) = c_\gamma$ coins from γ to η, the amount that A could pay D is increased to $min(b_2 + b_1, c_2 + c_\gamma)$. Correspondingly, the amount that D can pay A increase from $min(c_1, b_1)$ to $min(c_1 + c_2, b_1 + b_2)$. To sum up, with the same amount of deposits in channels, a path with bind facilitates more payments. In other words, users can bear the same amount of payments with fewer deposits.

2) Payment via a hub: Users could conduct payments through the payment hub. Here we focus on its typical and important scenarios [19] such as point-of-sale payments and retail shopping. Serving as one solution, the payment hub categorizes connected users into consumers and merchants. Figure 5 is a simplified situation, composed by consumer C, hub H, and merchants M_1 and M_2. The payment flow is unidirectional, from C to M_1 and M_2. C escrows c coins to β,

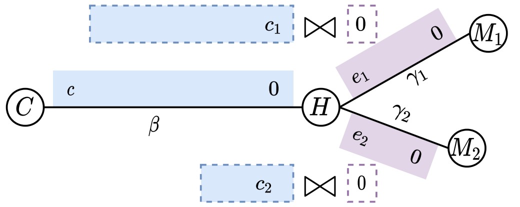


Fig. 5: Extension in payment hub network. C escrows c coins in β, where c_1 coins are for payments to M_1 and c_2 coins are for M_2. Out of the c coins, H binds c_1 coins to γ_1, and c_2 coins to γ_2. To deliver these payments, H escrows e_1 coins to γ_1 and e_2 coins to γ_2, where e_1 and e_2 are the maximum amount of a single payment to M_1 and M_2, respectively.

where a total of c_1 coins are used to pay M_1 and c_2 coins are for M_2. Each time C initiates a payment to the merchant, say M_1, H receives the coins in β and sends the same amount of coins to M_1 in γ_1. To accommodate these payments, H needs to escrow c_1 coins in γ_1 and c_2 coins in γ_2. In summary, for each user in the network, H needs to escrow the same amount of coins in the network.

With binds between these channels, the required deposits of H can be sharply decreased. H binds c_1 coins of β to γ_1 and c_2 coins to γ_2. At the same time, H escrows e_1 coins to γ_1 and e_2 coins to γ_2, where e_1 and e_2 are the maximum amount of a single payment to M_1 and M_2, respectively. Once a payment from C to a merchant, say M_1, completes, H shifts the received coins in β to γ_1. In this way, there are always e_1 coins in γ_1, which are sufficient for further payments. The deposits H needs to escrow in the network are decreased from c to $e_1 + e_2$. The above solution can be naturally extended to cases with multiple customers and merchants connecting to the same hub. When the number of customers and merchants is large, such as thousands or tens of thousands, the bind is more effective in reducing H's deposits in the network.

It is worth noting that the above analysis also demonstrates that single-channel users in Shaduf, e.g. A in Figure 4 and C in Figure 5, could help others, i.e. B and H, to achieve coin shift. This is attractive since the former constitutes a large portion of LN users, e.g. 45 percent in the snapshot.

IV. Formal Definition of Shaduf

A. The Security Model

We formalize Shaduf utilizing the universal composability (UC) [5] framework and deploy the version with global setup (GUC) [6]. The model is defined over a fixed set of parties $\mathcal{P} = \{P_1, ..., P_n\}$, taking the underlying blockchain as a global ideal functionality $\mathcal{L}$ with maximal blockchain delay Δ. It specifies two worlds, the real world and the ideal world. In the former world, the Shaduf protocol is executed by parties, facing an adversary $\mathcal{A}$ and interacting with the contract functionality $\mathcal{F}$. In the latter one, the idealized functionality $\mathcal{F}$ is executed through interactions with parties and the simulator $\mathcal{S}$ which simulates behaviors of the adversary. All parties receive inputs from the environment $\mathcal{Z}$ and send outputs to it. Security is defined as the computational indistinguishability between the outputs of $\mathcal{Z}$ running in the real world and in the ideal world.

1) Adversary model: We consider an adversary $\mathcal{A}$ who can corrupt arbitrary parties at the beginning of the protocol execution. Corruption means that $\mathcal{A}$ gets all internal states of corrupted parties and takes full control of them.

2) Security goals: Motivated by [8], [17], our main goal is to guarantee the balance security of honest users, i.e., honest users would not lose money. Intuitively, it includes two parts. Firstly, the channel opening and state update are only valid when both channel users agree. The channel binding and bind state update also need the approval of three users in two bound channels. Secondly, it requires that the channel unbinding and closing could be completed within a reasonable time, and when they are settled, the shifted coins between bound channels and refunded balances from a channel is the latest in honest users' view. Properties to be achieved are listed as follows.

Consensus on channel opening and channel update. Within a channel, the opening and state update can only be executed when both users agree. Channel opening takes time $O(\Delta)$ to achieve the agreement. When both users are honest, time spent on channel state update is constant.

Consensus on channel binding and bind update. Between two channels, the binding and state update takes effect only when all involved three users agree. Channel binding takes time $O(\Delta)$. Bind state update takes constant time that is independent of the blockchain delay when all users behave honestly.

Guaranteed channel unbinding. For two bound channels, any of the three users can request to unbind and settle the coin shift between them. The process is guaranteed to be completed within time $O(\Delta)$ after the request.

Guaranteed channel closing. Any user in a channel can request to close the channel at any time. The channel is closed in $O(\Delta)$ time once the request is made.

Guaranteed balance payout. The channel unbinding and closing are settled following the latest states from the view of honest users.

3) Communication network: We assume a synchronous communication network and each party is aware of the current time, which is measured in rounds. The delivery of a message between two parties takes 1 round. It means that one message sent in round r reaches the recipient in the beginning of round $r+1$. In addition, parties communicate via authenticated channels where the recipient can confirm the message source. The adversary can see the message contents but cannot modify or drop the messages. Other communications, e.g., with the environment, take 0 rounds. For simplicity, computations also take 0 rounds.

4) Ledger and contract functionalities: We take the formalization of the underlying blockchain from [8]. It is formalized as a global functionality $\mathcal{L}$ with respect to the blockchain delay Δ, denoted as $\mathcal{L}(\Delta)$. The functionality is shown below,

The value x_i in tuple $(x_1, ..., x_n)$ recorded in the ledger indicates P_i's balance and can be accessed by all parties, the adversary, and the environment. The "add" and "remove" operations are performed by the ideal functionality $\mathcal{F}$ in the ideal world, and the contract functionality $\mathcal{C}$ in the real world.

Ledger Functionality

Ledger initialization: Upon receiving $(x_1, ..., x_n) \in \mathbb{R}^n_{\geq 0}$ from the environment $\mathcal{Z}$, store the tuple.

Adding coins: Upon receiving (`add`, P_i, y), if $P_i \in \mathcal{P}$ and $y \in \mathbb{R}_{\geq 0}$ then set $x_i := x_i + y$, else do nothing.

Removing coins: Upon receiving (`remove`, P_i, y), if $P_i \in \mathcal{P}$ and $y \in \mathbb{R}_{\geq 0}$ and $x_i \geq y$ then set $x_i := x_i - y$, else do nothing.

We formalize the contract functionality $\mathcal{C}$ maintaining contract instances, leading to $\mathcal{C}$-hybrid real world. Each contract is identified by its corresponding channel and $\mathcal{C}(\beta)$ denotes the contract of channel β. Receiving messages from parties, $\mathcal{C}$ accesses the ledger according to predefined procedures.

5) Security definition: Let λ be the security parameter. With respect to the global ledger $\mathcal{L}(\Delta)$, denote the output of the environment $\mathcal{Z}$ when interacting with the adversary $\mathcal{A}$ and the protocol π in the $\mathcal{C}$-hybrid world as $\text{EXEC}^{\mathcal{L}(\Delta),\mathcal{C}}_{\pi,\mathcal{Z},\mathcal{A}}(\lambda)$, and $\mathcal{Z}$'s output when interacting with the ideal functionality $\mathcal{F}$ and the simulator $\mathcal{S}$ as $\text{EXEC}^{\mathcal{L}(\Delta)}_{\mathcal{F},\mathcal{Z},\mathcal{S}}(\lambda)$. The security is formally defined as follows.

Definition 1: Protocol π running in the $\mathcal{C}$-hybrid world UC-realizes the ideal functionality $\mathcal{F}$ with respect to a global ledger $\mathcal{L}(\Delta)$, if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that

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

where $\approx$ denotes the computational indistinguishability.

B. Notation

The following notations are used to define the attributes of the channel and the bind, and the communication between parties. They are also used in the real world protocol description.

1) Channel: A payment channel β is defined by the attribute tuple ($\beta.\mathsf{id}$, $\beta.\mathsf{users}$, $\beta.\mathsf{balance}$, $\beta.\mathsf{fund}$, $\beta.\mathsf{bind}$, $\beta.\mathsf{allot}$). Regarding the attributes, $\beta.\mathsf{id} \in \{0,1\}^*$ is the channel identifier, $\beta.\mathsf{users}$ = {$\beta.\mathsf{Alice}$, $\beta.\mathsf{Bob}$} is the set of channel users, $\beta.\mathsf{balance} : \beta.\mathsf{users} \rightarrow \mathbb{R}^{\geq 0}$ denotes the balances of users, $\beta.\mathsf{fund} \in \mathbb{R}^{\geq 0}$ is the deposits in the channel, $\beta.\mathsf{bind}$ records all binds of the channel, and $\beta.\mathsf{allot}$ is the remaining amount of coins for binding other channels and is initialized to be $\beta.\mathsf{fund}$. In addition, we define the function $\beta.\mathsf{other\text{-}user} : \beta.\mathsf{users} \rightarrow \beta.\mathsf{users}$ which maps one user to another one. Specifically, $\beta.\mathsf{other\text{-}user}$ ($\beta.\mathsf{Alice}$) = $\beta.\mathsf{Bob}$ and $\beta.\mathsf{other\text{-}user}$($\beta.\mathsf{Bob}$) = $\beta.\mathsf{Alice}$.

A payment θ in the channel is defined as $\beta.\mathsf{users} \rightarrow \mathbb{R}$ and $\theta(\beta.\mathsf{Alice}) + \theta(\beta.\mathsf{Bob}) = 0$. Applying payment θ to the current channel balance. i.e., $\beta.\mathsf{balance} + \theta$, derives balances of users after the payment.

2) Bind: A bind Φ is defined by the attribute tuple ($\Phi.\mathsf{id}$, $\Phi.\mathsf{users}$, $\Phi.\mathsf{channels}$, $\Phi.\mathsf{shift}$, $\Phi.\mathsf{reserve}$). About the attributes, $\Phi.\mathsf{id} \in \{0,1\}^*$ is the bind identifier, and $\Phi.\mathsf{users} = (\Phi.\mathsf{Lara}, \Phi.\mathsf{Irene}, \Phi.\mathsf{Rock})$ is the set of users, where $\Phi.\mathsf{Irene}$ is the intermediate. $\Phi.\mathsf{channels} = (\Phi.\mathsf{c_a}, \Phi.\mathsf{c_p})$ is the set of two bound channels, and $\Phi.\mathsf{c_a}$ consists of users $\Phi.\mathsf{Lara}$ and $\Phi.\mathsf{Irene}$, while $\Phi.\mathsf{c_p}$ takes $\Phi.\mathsf{Irene}$ and $\Phi.\mathsf{Rock}$ as its users. $\Phi.\mathsf{shift} : \Phi.\mathsf{channels} \rightarrow \mathbb{R}$ denotes the total shifted coins between the channels and $\Phi.\mathsf{shift}(\Phi.\mathsf{c_a}) + \Phi.\mathsf{shift}(\Phi.\mathsf{c_p}) = 0$. For example, if $\Phi.\mathsf{shift}(\Phi.\mathsf{c_a}) = c > 0$, it indicates c coins are shifted to $\Phi.\mathsf{c_a}$ from $\Phi.\mathsf{c_p}$. $\Phi.\mathsf{reserve} : \Phi.\mathsf{channels} \rightarrow \mathbb{R}^{>0}$ indicates the upper limit of coin shift in each channel. When $\Phi.\mathsf{reserve}(\Phi.\mathsf{c_a}) = c$, it means that at most c coins can be shifted from $\Phi.\mathsf{c_a}$ to $\Phi.\mathsf{c_p}$. Furthermore, we define $\Phi.\mathsf{end\text{-}users} = \{\Phi.\mathsf{Lara}, \Phi.\mathsf{Rock}\}$ as the set of end users, and the function $\Phi.\mathsf{other\text{-}users}$ which maps one user to other two users. To cite an example, $\Phi.\mathsf{other\text{-}users}(\Phi.\mathsf{Lara}) = \{\Phi.\mathsf{Irene}, \Phi.\mathsf{Rock}\}$.

One coin shift $\tilde{\theta}$ between two channels is defined as $\Phi.\mathsf{channels} \rightarrow \mathbb{R}$ and $\tilde{\theta}(\Phi.\mathsf{c_a}) + \tilde{\theta}(\Phi.\mathsf{c_p}) = 0$. When $\tilde{\theta}$ is executed, the resulting coin shift is updated to $\Phi.\mathsf{shift} + \tilde{\theta}$.

3) Communication: We write the event that "send message m to P in round r" as "$m \overset{r}{\hookrightarrow} P$" and "receive message m from P in round r" as "$m \overset{r}{\hookleftarrow} P$", where P could be a party or a contract. Especially, when P is a set of parties, the above presentations denote sending or receiving the message to or from each party in the set. Moreover, when the time is termed as $r \leq r_1$, it means that the event is happened in round r which is decided by the adversary but r_1 is the upper bound.

C. Ideal Functionality of Shaduf

The ideal functionality $\mathcal{F}$ communicates with parties in the set $\mathcal{P}$, the simulator $\mathcal{S}$, and the ledger $\mathcal{L}$. All existing channels and bindings are recorded in the set Γ and Λ, respectively. Since Shaduf does not concern the privacy issue, $\mathcal{F}$ forwards to $\mathcal{S}$ the correct messages received from parties and messages sent to parties. To simplify the functionality description, we omit some of the functionality's checks on the correctness of received messages here. These checks are natural, such as checking that both channel users have sufficient coins on the ledger $\mathcal{L}$ for the channel deposit in the case of channel opening. We list these omitted checks in Appendix A.

There are 6 procedures in $\mathcal{F}$, as shown in Figure 6. Each procedure is triggered by one message from parties, including the procedure name and an identifier referring to the channel or the bind. Procedure (A) `Open` handles the opening of channel β when triggered by $\beta.\mathsf{Alice}$. (B) `Channel Update` performs the state update, or in other words, the payment, within the channel when receiving a request from one user and confirmed by another one. (C) `Bind` processes the bind of two channels and is triggered by the intermediate. (D) `Bind Update` executes the coin shift between two bound channels when receiving requests from the intermediate and agreement from the end users. (E) `Unbind` resolves the bind of two channels when triggered by any user in them and completes the final coin shift between them. (F) `Close` is triggered by any channel user and refunds balances on the blockchain.

Now we analyze the achievement of desired security goals.

Consensus on channel opening and channel update. A channel can only be opened when Alice applies and Bob approves it. This means that two users have achieved agreement on channel opening. From the time interval between the message (`opened`) and the opening request, it is easy to see the opening takes time $O(\Delta)$. Similarly, the channel state update is completed when both users agree. When both users are honest, it takes 2 rounds to confirm the result.

(A) **Open**

Upon $(\texttt{open}, \beta) \overset{t}{\hookleftarrow} \beta.\mathsf{Alice}$, $(\texttt{remove}, \beta.\mathsf{Alice}, \beta.\mathsf{balance}(\beta.\mathsf{Alice})) \xhookrightarrow{t_1 \le t+\Delta} \mathcal{L}$, $(\texttt{opening}, \beta) \xhookrightarrow{t_1} \beta.\mathsf{Bob}$. If $(\texttt{open}) \overset{t_1}{\hookleftarrow} \beta.\mathsf{Bob}$, send $(\texttt{remove}, \beta.\mathsf{Bob}, \beta.\mathsf{balance}(\beta.\mathsf{Bob})) \xhookrightarrow{t_2 \le t_1+\Delta} \mathcal{L}$, $(\texttt{opened}) \xhookrightarrow{t_2} \beta.\mathsf{users}$, set $\Gamma(\beta.\mathsf{id}) := \beta$, and stop. Otherwise, upon $(\texttt{refund}, \beta) \xhookleftarrow{t_3 > t_1+\Delta} \beta.\mathsf{Alice}$, send $(\texttt{add}, \beta.\mathsf{Alice}, \beta.\mathsf{balance}(\beta.\mathsf{Alice})) \xhookrightarrow{t_4 \le t_3+\Delta} \mathcal{L}$, and stop.

(B) **Channel Update**

Upon $(\texttt{chan-update}, id, \theta) \overset{t}{\hookleftarrow} P \in \beta.\mathsf{users}$, let $\beta := \Gamma(id)$, and send $(\texttt{chan-update-req}, id, \theta) \xhookrightarrow{t+1} Q$, where $Q := \beta.\mathsf{other\text{-}user}(P)$. If $(\texttt{chan-update-ok}) \xhookleftarrow{t+1} Q$, $\beta.\mathsf{balance} := \beta.\mathsf{balance} + \theta$, $(\texttt{chan-updated}) \xhookrightarrow{t+2} P$, and stop. Else, if P is honest, denote θ as β's pending update, execute procedure (F), and stop. Otherwise, stop.

(C) **Bind**

Upon $(\texttt{bind}, \Phi) \overset{t}{\hookleftarrow} \Phi.\mathsf{Irene}$, if $\Phi.\mathsf{Irene}$ is honest, $(\texttt{bind-req}, \Phi) \xhookrightarrow{t+1} \Phi.\mathsf{end\text{-}users}$. Otherwise, for $P \in \Phi.\mathsf{end\text{-}users}$: if $(\texttt{send-req}, P) \overset{t}{\hookleftarrow} \mathcal{S}$, $(\texttt{bind-req}, \Phi) \xhookrightarrow{t+1} P$. Do the following.

1) If $(\texttt{bind-ok}) \xhookleftarrow{t+1} \Phi.\mathsf{end\text{-}users}$, $(\texttt{bind-confirm}) \xhookrightarrow{t+2} \Phi.\mathsf{Irene}$. Else, stop.
2) If $(\texttt{bind-confirmed}) \xhookleftarrow{t+2} \Phi.\mathsf{Irene}$, $(\texttt{bound}) \xhookrightarrow{t_1 \le t+2+\Delta} \Phi.\mathsf{users}$, set $\Lambda(\Phi.\mathsf{id}) := \Phi$, for $\gamma \in \Phi.\mathsf{channels}$: add Φ to $\gamma.\mathsf{bind}$, $\gamma.\mathsf{allot} := \gamma.\mathsf{allot} - \Phi.\mathsf{reserve}(\gamma)$, and stop. Else, $(\texttt{not-bound}) \xhookrightarrow{t+2+\Delta} \Phi.\mathsf{end\text{-}users}$, and stop.

(D) **Bind Update**

Upon $(\texttt{bind-update}, \widetilde{id}, \tilde{\theta}) \overset{t}{\hookleftarrow} I$, let $\Phi := \Lambda(\widetilde{id})$. If I is honest, $(\texttt{bind-update-req}, \widetilde{id}, \tilde{\theta}) \xhookrightarrow{t+1} \Phi.\mathsf{end\text{-}users}$. Otherwise, for $P \in \Phi.\mathsf{end\text{-}users}$: if $(\texttt{send-req}, P) \overset{t}{\hookleftarrow} \mathcal{S}$, $(\texttt{bind-update-req}, \widetilde{id}, \tilde{\theta}) \xhookrightarrow{t+1} P$. Do the following,

1) If $(\texttt{bind-update-ok}) \xhookleftarrow{t_1 \le t+2} \Phi.\mathsf{end\text{-}users}$, $\Phi.\mathsf{shift} := \Phi.\mathsf{shift} + \tilde{\theta}$, $(\texttt{bind-updated}) \xhookrightarrow{t+3} \Phi.\mathsf{users}$. For $\gamma \in \Phi.\mathsf{channels}$: $\gamma.\mathsf{balance}(I) := \gamma.\mathsf{balance}(I) + \tilde{\theta}(\gamma)$. Else, if I is honest, or $(\texttt{bind-update-ok}) \xhookleftarrow{t+1} P \in \Phi.\mathsf{end\text{-}users}$ and P is honest, denote $\tilde{\theta}$ as Φ's pending update, go to procedure (E), and stop. Otherwise, stop.
2) For $\gamma \in \Phi.\mathsf{channels}$: if $(\texttt{bind-chan-not-updated}, \gamma.\mathsf{id}) \xhookleftarrow{t_2 \le t+4} P \in \gamma.\mathsf{users}$ and P is corrupted, $(\texttt{bind-chan-not-updated}, \gamma.\mathsf{id}) \xhookrightarrow{t_2+1} Q$ where $Q := \gamma.\mathsf{other\text{-}user}(P)$, execute procedure (F), and stop. Else, stop.

(E) **Unbind**

Upon $(\texttt{unbind}, \widetilde{id}) \overset{t}{\hookleftarrow} P$ (let $\Phi := \Lambda(\widetilde{id})$), or be invoked by other procedures (let t be the current round), in round $t_1 \le t + \Delta$, if Φ's pending update $\tilde{\theta}$ is confirmed, $\Phi.\mathsf{shift} := \Phi.\mathsf{shift} + \tilde{\theta}$, and for $\gamma \in \Phi.\mathsf{channels}$: $\gamma.\mathsf{balance}(\Phi.\mathsf{Irene}) := \gamma.\mathsf{balance}(\Phi.\mathsf{Irene}) + \tilde{\theta}(\gamma)$. Send $(\texttt{unbound}) \xhookrightarrow{t_2 \le t_1+2\Delta} \Phi.\mathsf{users}$, and set $\Lambda(\Phi.\mathsf{id}) :=\perp$. For $\gamma \in \Phi.\mathsf{channels}$, do the following,

1) $\gamma.\mathsf{fund} := \gamma.\mathsf{fund} + \Phi.\mathsf{shift}(\gamma)$, $\gamma.\mathsf{allot} := \gamma.\mathsf{allot} + \Phi.\mathsf{shift}(\gamma) + \Phi.\mathsf{reserve}(\gamma)$, and remove Φ from $\gamma.\mathsf{bind}$.
2) If $(\texttt{bind-chan-not-updated}, \gamma.\mathsf{id}) \xhookleftarrow{t_3 \le t_2+1} P \in \gamma.\mathsf{users}$ and P is corrupted, send $(\texttt{bind-chan-not-updated}, \gamma.\mathsf{id}) \xhookrightarrow{t_3+1} Q$ where $Q := \gamma.\mathsf{other\text{-}user}(P)$, execute procedure (F) and stop. Else, stop.

(F) **Close**

Upon $(\texttt{close}, id) \overset{t}{\hookleftarrow} P$ (let $\beta := \Gamma(id)$), or be invoked by other procedures (let t be the current round), in round $t_1 \le t + \Delta$, if β's pending update θ is confirmed, $\beta.\mathsf{balance} := \beta.\mathsf{balance} + \theta$. Do the following,

1) If $\beta.\mathsf{bind} = \emptyset$, $(\texttt{add}, \beta.\mathsf{Alice}, \beta.\mathsf{balance}(\beta.\mathsf{Alice})) \xhookrightarrow{t_2 \le t_1+2\Delta} \mathcal{L}$, $(\texttt{add}, \beta.\mathsf{Bob}, \beta.\mathsf{balance}(\beta.\mathsf{Bob})) \xhookrightarrow{t_2} \mathcal{L}$, $(\texttt{closed}) \xhookrightarrow{t_2} \beta.\mathsf{users}$, set $\Gamma(\beta.\mathsf{id}) :=\perp$, and stop.
2) Otherwise, for each bind in $\beta.\mathsf{bind}$: execute procedure (E). Let t_3 be the current time. $(\texttt{add}, \beta.\mathsf{Alice}, \beta.\mathsf{balance}(\beta.\mathsf{Alice})) \xhookrightarrow{t_4 \le t_3+\Delta} \mathcal{L}$, $(\texttt{add}, \beta.\mathsf{Bob}, \beta.\mathsf{balance}(\beta.\mathsf{Bob})) \xhookrightarrow{t_4} \mathcal{L}$, $(\texttt{closed}) \xhookrightarrow{t_4} \beta.\mathsf{users}$, set $\Gamma(\beta.\mathsf{id}) :=\perp$, and stop.

Fig. 6: Ideal functionality $\mathcal{F}$

Consensus on channel binding and bind update. The bind of two channels is initiated by the intermediate and confirmed by the end users. Channels are bound only when they all agree, taking $O(\Delta)$ time. Likewise, the bind state is updated when all users confirm. It takes 3 rounds when all users behave honestly.

Guaranteed channel unbinding. The channel unbinding can be triggered by any user of the two bound channels, and it is guaranteed to be completed within 3Δ time.

Guaranteed channel closing. The channel closing can be initiated by any user of the channel. When the channel is not bound to other channels, the time needed to close the channel is at most 3Δ. Otherwise, the time is at most 5Δ, among which 3Δ is used to unbind other channels.

Guaranteed balance payout. When two channels are unbound, the coin shift is settled according to its latest bind state ($\Phi.\mathsf{shift}$ in Figure 6 (E)). Similarly, when the channel is closed, the latest balance allocation between two end users is put on the blockchain ($\beta.\mathsf{balance}$ in Figure 6 (F)).

V. Detailed Description of Shaduf

In this section, we give a detailed description of Shaduf protocol. Before the description, we first introduce the supple-

(A) **Open**

1) Upon (`open`, β) $\overset{t}{\hookleftarrow} \mathcal{Z}$, β.Alice sends (`open`, β) $\overset{t}{\hookrightarrow} \mathcal{C}(\beta)$, and goes to step 3.
2) Upon (`opening`, β) $\overset{\tau}{\hookleftarrow} \mathcal{C}(\beta)$, β.Bob sends (`opening`, β) $\overset{\tau}{\hookrightarrow} \mathcal{Z}$. If (`open`) $\overset{\tau}{\hookleftarrow} \mathcal{Z}$, he sends (`open`) $\overset{\tau}{\hookrightarrow} \mathcal{C}(\beta)$, outputs (`opened`) upon receiving it from $\mathcal{C}(\beta)$, sets $\Gamma^B(\beta.\mathsf{id}) := \beta$, and waits for messages concerning β. Otherwise, he stops.
3) If (`opened`) $\xhookleftarrow{t_1 \leq t+2\Delta} \mathcal{C}(\beta)$, β.Alice sets $\Gamma^A(\beta.\mathsf{id}) := \beta$, outputs (`opened`), and waits for messages concerning β. Otherwise, upon receiving (`refund`, β) from $\mathcal{Z}$, she sends (`timeout`) to $\mathcal{C}(\beta)$, and stops.

(B) **Channel Update**

1) Upon (`chan-update`, id, θ) $\overset{t}{\hookleftarrow} \mathcal{Z}$, P generates signature σ_P on $msg = (\beta^P.\mathsf{version} + 1, \beta^P.\mathsf{balance} + \theta, \beta^P.\mathsf{biList})$, sends (`chan-update`, id, msg, σ_P) $\overset{t}{\hookrightarrow} Q$, and goes to step 3.
2) Upon (`chan-update`, $id, msg = (ver, \alpha, biList)$, σ_P) $\overset{\tau}{\hookleftarrow} P$, Q sends (`chan-update-req`, id, $\alpha - \beta^Q.\mathsf{balance}$) $\overset{\tau}{\hookrightarrow} \mathcal{Z}$. If (`chan-update-ok`) $\overset{\tau}{\hookleftarrow} \mathcal{Z}$, Q generates signature σ_Q on msg, sends (`chan-update`, id, σ_Q) $\overset{\tau}{\hookrightarrow} P$, updates $\beta^Q.\mathsf{state}$, and stops. Otherwise, Q stops.
3) If (`chan-update`, id, σ_Q) $\xhookleftarrow{t+2} Q$ where σ_Q is Q's signature on msg, P outputs (`chan-updated`), updates $\beta^P.\mathsf{state}$, and stops. Else, P denotes θ as β's pending update, initiates channel closing in procedure (F), and stops.

(C) **Bind**

1) Upon (`bind`, Φ) $\overset{t}{\hookleftarrow} \mathcal{Z}$, I generates signature σ_I on (Φ, t), sends (`bind`, Φ, t, σ_I) $\overset{t}{\hookrightarrow} \Phi.\mathsf{end\text{-}users}$, and goes to step 3.
2) Upon (`bind`, $\Phi, \tau - 1, \sigma_I$) $\overset{\tau}{\hookleftarrow} I$ where σ_I is I's signature on $msg = (\Phi, \tau - 1)$, $P \in \Phi.\mathsf{end\text{-}users}$ sends (`bind-req`, Φ) $\overset{\tau}{\hookrightarrow} \mathcal{Z}$. If (`bind-ok`) $\overset{\tau}{\hookleftarrow} \mathcal{Z}$, P generates signature σ_P on msg, (`bind`, σ_P) $\overset{\tau}{\hookrightarrow} I$, and goes to step 4. Else, P stops.
3) If I receives the end users' signatures on (Φ, t) in $t+2$, I sends (`bind-confirm`) $\xhookrightarrow{t+2} \mathcal{Z}$ and if (`bind-confirmed`) $\xhookleftarrow{t+2} \mathcal{Z}$, I sends $(\Phi, t, \Sigma) \xhookrightarrow{t+2} \mathcal{C}(\Phi.\mathsf{c_a})$ where Σ is the set of signatures, outputs (`bound`) upon receiving it from $\mathcal{C}(\Phi.\mathsf{c_a})$, sets $\Lambda^I(\Phi.\mathsf{id}) := \Phi$, and for $\gamma \in \Phi.\mathsf{channels}$: add Φ to $\gamma^I.\mathsf{bind}$, set $\gamma.\mathsf{allot} := \gamma.\mathsf{allot} - \Phi.\mathsf{reserve}(\gamma)$, and stop. Otherwise, I stops.
4) Denote P in channel $\gamma \in \Phi.\mathsf{channels}$. If (`bound`) $\xhookleftarrow{\tau_1 \leq \tau+1+\Delta} \mathcal{C}(\Phi.\mathsf{c_a})$, P sets $\Lambda^P(\Phi.\mathsf{id}) := \Phi$, adds Φ to $\gamma^P.\mathsf{bind}$, sets $\gamma.\mathsf{allot} := \gamma.\mathsf{allot} - \Phi.\mathsf{reserve}(\gamma)$, outputs (`bound`), and stops. Otherwise, P outputs (`not-bound`) and stops.

Fig. 7: Procedures (A) Open, (B) Channel Update, and (C) Bind.

mentary notations on bind and channel. The protocol is shown in Figure 7 and 8, and the contract functionality is shown in Figure 9. Finally, we provide the security theorem.

A. Notation

Recall in Section III-C, parties maintain states of the bind and the channel to perform off-chain coin shifts and payments. In the formal description, the states are presented as follows,

1) Bind: The state of bind Φ is defined as $\Phi.\mathsf{state}$ = ($\Phi.\mathsf{version}$, $\Phi.\mathsf{shift}$, $\Phi.\mathsf{laShift}$, $\Phi.\mathsf{sig}$), where $\Phi.\mathsf{version}$ is the shift counter and increments by 1 each shift, $\Phi.\mathsf{shift}$ is the total coin shift and $\Phi.\mathsf{laShift}$ is the latest coin shift between the two bound channels, $\Phi.\mathsf{sig}$ collects signatures from the three users on this state. The state is called valid and can be submitted to the blockchain when $\Phi.\mathsf{sig}$ includes the signatures of all three users. We denote the latest valid state of bind Φ that party P is aware of as $\Phi^P.\mathsf{state}$.

2) Channel: The state of channel β is represented as $\beta.\mathsf{state} = (\beta.\mathsf{version}, \beta.\mathsf{balance}, \beta.\mathsf{biList}, \beta.\mathsf{sig})$. Among the attributes, $\beta.\mathsf{version}$ is the counter of state update and increments by 1 each update, $\beta.\mathsf{balance}$ is balance allocation between two users, $\beta.\mathsf{biList} : \{0,1\}^* \to \mathbb{N}^*$ maps the identifier of a bind to its version, and $\beta.\mathsf{sig}$ records signatures of channel users on the state. Similarly, the state is valid when $\beta.\mathsf{sig}$ contains signatures from both the users. And $\beta^P.\mathsf{state}$ represents the latest valid channel state from P's view.

B. Procedures of Shaduf

We assume that each party $P \in \mathcal{P}$ maintains all existing channels and bindings in set Γ^P and Λ^P, respectively. Furthermore, to simplify the protocol description, we extract the checks that honest users make on received messages and show them in Appendix B. We start with describing the procedure of channel opening as shown in Figure 7(A), with Figure 9(A) presenting contract actions. β.Alice sends the opening request to the contract instance $\mathcal{C}(\beta)$, which informs β.Bob of the opening event. If β.Bob confirms it within Δ time, the contract marks β as opened and informs (`opened`) to them (Step (1), Figure 9(A)). Otherwise, β.Alice sends (`timeout`) to the contract, which returns back the coins (Step (2), Figure 9(A)).

The channel state update is performed off-chain between the channel users. When user P receives the instruction to update the channel state, P increments the update counter and the balance allocation, and sends the updated state to the other user, say Q, along with the signature (Step (1), Figure 7(B)). Q verifies the correctness of the received message and asks the environment whether the update is supported. If it is, Q replies with the signature on the updated state, which becomes valid then (Step (2), Figure 7(B)). If Q does not respond, P is not sure whether the state is updated. Then P marks the update as pending and initiates channel closing, from which the result of update is derived.

The procedure of bind is shown in Figure 7(C) and the contract operation appears in Steps (1-2) of Figure 9(B). To

(D) **Bind Update**

1) Upon (`bind-update`, $\widetilde{id}$, $\tilde{\theta}$) $\overset{t}{\hookleftarrow}$ $\mathcal{Z}$, I lets $\Phi := \Lambda^I(\widetilde{id})$, generates signature σ_I on *msg* $= (\Phi^I.\mathsf{version}+1, \Phi^I.\mathsf{shift}+\tilde{\theta}, \tilde{\theta})$, sends (`bind-update`, $\widetilde{id}, msg, \sigma_I$) $\overset{t}{\hookrightarrow}$ $\Phi.\mathsf{end\text{-}users}$, and goes to step 3.
2) Upon (`bind-update`, $\widetilde{id}, msg = (ver, shift, \tilde{\theta}), \sigma_I$) $\overset{\tau}{\hookleftarrow}$ I, $P \in \Phi.\mathsf{end\text{-}users}$ lets $\Phi := \Lambda^P(\widetilde{id})$, sends (`bind-update-req`, $\widetilde{id}$, $\tilde{\theta}$) $\overset{\tau}{\hookrightarrow}$ $\mathcal{Z}$. If (`bind-update-ok`) $\overset{\tau}{\hookleftarrow}$ $\mathcal{Z}$, P generates signature σ_P on *msg*, (`bind-update`, $\widetilde{id}, \sigma_P$) $\overset{\tau}{\hookrightarrow}$ $\Phi.\mathsf{other\text{-}users}(P)$, and goes to step 4. Otherwise, P stops.
3) Upon (`bind-update`, $\widetilde{id}, \sigma_P$) $\xhookleftarrow{t+2}$ $P \in \Phi.\mathsf{end\text{-}users}$ where σ_P is P's signature on *msg*, I forwards the message to the other end user. In round $t+3$, if the end users' signatures are collected, I outputs (`bind-updated`), updates $\Phi^I.\mathsf{state}$, and for $\gamma \in \Phi.\mathsf{channels}$: goes to step 5. Otherwise, I denotes $\tilde{\theta}$ as Φ's pending update, initiates unbinding in the procedure (E) and stops.
4) If (`bind-update`, $\widetilde{id}, \sigma_Q$) $\xhookleftarrow{\tau+1}$ Q where Q is the other end user and σ_Q is Q's signature on *msg*, P forwards it to I. In round $\tau+2$, if P has collected Q's signature, P outputs (`bind-updated`), updates $\Phi^P.\mathsf{state}$, and in P's channel γ: goes to step 6. Otherwise, P denotes $\tilde{\theta}$ as Φ's pending update, initiates unbinding in procedure (E), and stops.
5) Denote the current round as t_1. I updates her balance by $\tilde{\theta}(\gamma)$ and denotes the updated channel balance as $\gamma^I.\mathsf{balance}'$, adds 1 to $\gamma^I.\mathsf{biList}(\widetilde{id})$ (sets to 1 if $\widetilde{id}$ is not in the list) and denotes it as $\gamma^I.\mathsf{biList}'$, generates signature σ_I on *msg* $= (\gamma^I.\mathsf{version}+1, \gamma^I.\mathsf{balance}', \gamma^I.\mathsf{biList}')$, and sends it to the other channel user. If the other user's signature on *msg* is received, I updates $\beta^I.\mathsf{state}$ and stops. Otherwise, I outputs (`bind-chan-not-updated`), initiates channel closing in procedure (F), and stops.
6) Denote the current round as τ_1. P updates I's balance by $\tilde{\theta}(\gamma)$ and denotes the updated channel balance as $\gamma^P.\mathsf{balance}'$, adds 1 to $\gamma^P.\mathsf{biList}(\widetilde{id})$ (sets to 1 if $\widetilde{id}$ is not in the list) and denotes it as $\gamma^P.\mathsf{biList}'$. If P receives I's signature on *msg* $= (\gamma^P.\mathsf{version}+1, \gamma^P.\mathsf{balance}', \gamma^P.\mathsf{biList}')$, P sends the signature on *msg* to I, updates $\beta^P.\mathsf{state}$, and stops. Otherwise, P outputs (`bind-chan-not-updated`), initiates channel closing in procedure (F), and stops.

(E) **Unbind**

1) Upon (`unbind`, $\widetilde{id}$) $\overset{t}{\hookleftarrow}$ $\mathcal{Z}$, P lets $\Phi := \Lambda^P(\widetilde{id})$, sends (`unbind`, $\widetilde{id}, \Phi^P.\mathsf{state}$) $\overset{t}{\hookrightarrow}$ $\mathcal{C}(\Phi.\mathsf{c_a})$, waits for Δ time, and goes to step 3.
2) Upon (`unbinding`, $\widetilde{id}$) $\overset{\tau}{\hookleftarrow}$ $\mathcal{C}(\Phi.\mathsf{c_a})$, Q sends (`unbind`, $\widetilde{id}, \Phi^Q.\mathsf{state}$) $\overset{\tau}{\hookrightarrow}$ $\mathcal{C}(\Phi.\mathsf{c_a})$, and goes to step 3.
3) If not received (`unbound`) from $\mathcal{C}(\Phi.\mathsf{c_a})$ within Δ time, then send (`timeout`, $\widetilde{id}$) to $\mathcal{C}(\Phi.\mathsf{c_a})$. Upon receiving (`unbound`) from $\mathcal{C}(\Phi.\mathsf{c_a})$, output (`unbound`), set $\Lambda(\widetilde{id}) := \perp$, and for $\gamma \in \Phi.\mathsf{channels}$ where the party is in, proceed as follows,
   a) Remove Φ from $\gamma.\mathsf{bind}$, update $\gamma.\mathsf{fund}$ and $\gamma.\mathsf{allot}$ as the contract $\mathcal{C}(\gamma)$.
   b) If Φ's pending update is confirmed and γ has not been initiated closing, go to step (5) of procedure (D) if the party is I and step (6) otherwise. Stop the procedure.

(F) **Close**

1) Upon (`close`, id) $\overset{t}{\hookleftarrow}$ $\mathcal{Z}$, P lets $\beta := \Gamma^P(id)$, sends (`close`, $\beta^P.\mathsf{state}$) $\overset{t}{\hookrightarrow}$ $\mathcal{C}(\beta)$. If $\beta.\mathsf{bind} = \emptyset$, P waits for Δ time and goes to step 3. Otherwise, P starts procedure (E) for each bind in $\beta.\mathsf{bind}$. When all bindings are unbound, P goes to step 4.
2) Upon (`closing`) $\overset{\tau}{\hookleftarrow}$ $\mathcal{C}(\beta)$, Q sends (`close`, $\beta^Q.\mathsf{state}$) $\overset{\tau}{\hookrightarrow}$ $\mathcal{C}(\beta)$. If $\beta.\mathsf{bind} = \emptyset$, Q goes to step 3. Otherwise, Q executes procedure (E) for each bind in $\beta.\mathsf{bind}$. When all bindings are unbound, Q goes to step 4.
3) If the party received (`closed`) from $\mathcal{C}(\beta)$ within Δ time, output (`closed`), set $\Gamma(\beta.\mathsf{id}) := \perp$, and stop. Otherwise, send (`close-resolve`) to $\mathcal{C}(\beta)$ after Δ time, output (`closed`) upon receiving it from $\mathcal{C}(\beta)$, set $\Gamma(\beta.\mathsf{id}) := \perp$, and stop.
4) Send (`close-resolve`) to $\mathcal{C}(\beta)$, output (`closed`) upon receiving it from $\mathcal{C}(\beta)$, set $\Gamma(\beta.\mathsf{id}) := \perp$, and stop.

Fig. 8: Procedures (D) Bind Update, (E) Unbind, and (F) Close.

bind two channels, users need to declare it on the blockchain. It is required that the two contracts record the bind atomically. If one contract confirms the bind while the other one not, the eventual coin shift between the two channels cannot be settled since the contract can only send or receive coins with bound channels.[1] To guarantee the above cases would not happen, the setup of bind in one contract is triggered by another contract, which in turn records the bind only when the triggered contract returns confirmation. Specifically, users send the bind request to the trigger channel contract, which verifies the request and informs the triggered contract if correct (Step (1a), Figure 9(B)). The triggered contract checks the bind and replies confirmation if correct (Step (2), Figure 9(B)). After that, the trigger contract records the bind and informs users (Step (1a), Figure 9(B)). To specify the character of "trigger" and "triggered" between the two contracts, we fix the contract with a lower channel identifier value as "trigger" and the other one as "triggered". Their roles are the same when unbinding.

The request of bind Φ is submitted by the intermediate, who needs to collect signatures of the other two users on the request first (Step (1), Figure 7(C)). The request is in the form of (Φ, t), where t is the timestamp to avoid the replay attack.

[1] It is also required that the contracts are secure, which can handle the unbind request and deliver shifted coins to bound channels. It can be implemented by checking whether the contract is of permitted implementations via opcode EXTCODEHASH, see https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1052.md.

(A) The contract for channel opening

Upon (open, β) $\overset{\tau}{\hookleftarrow}$ β.Alice, ignore the message is both channel users' coins on the ledger $\mathcal{L}$. Otherwise, send (remove, β.Alice, β.balance(β.Alice)) $\overset{\tau}{\hookrightarrow}$ $\mathcal{L}$, (opening, β) $\overset{\tau}{\hookrightarrow}$ β.Bob, and wait for one of the following messages,

1) (open) $\xhookleftarrow{\tau_1 \leq \tau+\Delta}$ β.Bob: (remove, β.Bob, β.balance(β.Bob)) $\overset{\tau_1}{\hookrightarrow}$ $\mathcal{L}$, (opened) $\overset{\tau_1}{\hookrightarrow}$ β.users, and go to point (B).
2) (timeout) $\xhookleftarrow{\tau_2 > \tau+\Delta}$ β.Alice: (add, β.Alice, β.balance(β.Alice)) $\overset{\tau_2}{\hookrightarrow}$ $\mathcal{L}$, and close the contract.

(B) The contract execution of channel β

Wait for the following messages:

1) (bind, Φ, t, Σ) $\xhookleftarrow{\tau \leq t+2+\Delta}$ Φ.Irene where β is Φ.c$_a$ and Σ are signatures of Φ.users on (Φ, t) :
   a) If $\Phi \notin \beta$.bind and β.allot $\geq$ Φ.reserve(β), trigger (bind, Φ) of $\mathcal{C}(\Phi.\mathsf{c_p})$. If $\mathcal{C}(\Phi.\mathsf{c_p})$ returns (bound), add Φ to β.bind, β.allot := β.allot $-$ Φ.reserve(β), (bound) $\overset{\tau}{\hookrightarrow}$ Φ.users. /*binding validity verification*/
   b) Otherwise, ignore the message.
2) (bind, Φ) $\overset{\tau}{\hookleftarrow}$ $\mathcal{C}(\Phi.\mathsf{c_a})$: If $\Phi \notin \beta$.bind and β.allot $\geq$ Φ.reserve(β), add Φ to β.bind, β.allot := β.allot $-$ Φ.reserve(β), return (bound). Otherwise, ignore the message. /*binding validity verification*/
3) (unbind, $\widetilde{id}$, W) $\overset{\tau}{\hookleftarrow}$ P where $\widetilde{id}$ is the identifier of $\Phi \in \beta$.bind and β is Φ.c$_a$: set Φ.toResp := Φ.other-users(P), (unbinding, $\widetilde{id}$) $\overset{\tau}{\hookrightarrow}$ Φ.toResp. If W is a valid state, set Φ.state := W. Wait for the following messages,
   a) (unbind, $\widetilde{id}$, W') $\xhookleftarrow{\tau_1 \leq \tau+\Delta}$ P' where $P' \in \Phi$.toResp : Remove P' from Φ.toResp. If W' is a valid state and W'.version $>$ Φ.version, set Φ.state := W'. If Φ.toResp $= \emptyset$, proceed procedure (C), remove Φ from β.bind, store Φ.version and Φ.laShift, invoke (unbind, $\widetilde{id}$, Φ.state) of $C(\Phi.\mathsf{c_p})$, and (unbound) $\overset{\tau}{\hookrightarrow}$ Φ.users.
   b) (timeout, $\widetilde{id}$) after Δ time : proceed procedure (C), remove Φ from β.bind, store Φ.version and Φ.laShift, invoke (unbind, $\widetilde{id}$, Φ.state) of Φ.c$_p$, and send (unbound) to Φ.users.
4) (unbind, $\widetilde{id}$, Φ.state) $\overset{\tau}{\hookleftarrow}$ $C(\Phi.\mathsf{c_a})$: proceed procedure (C), remove Φ from β.bind, store Φ.version and Φ.laShift.
5) (close, M) $\overset{\tau}{\hookleftarrow}$ P : if M is a valid state, set β.state := M. Denote P' := β.other-user(P), (closing) $\overset{\tau}{\hookrightarrow}$ P', and wait for the following messages,
   a) (close, M') $\xhookleftarrow{\tau_1 \leq \tau+\Delta}$ P' : if M' is a valid state and M'.version $>$ β.version, set β.state := M'. If β.bind $= \emptyset$, go to procedure (D).
   b) (close-resolve) after Δ time : If β.bind $= \emptyset$, go to procedure (D).

(C) Subroutine for settling Φ's coin shift

Let γ be the channel bound to β. Distinguish the following cases according to Φ.shift(β), denoted as *amt*,

1) If *amt* > 0 and *amt* $\leq$ Φ.reserve(γ), or *amt* < 0 and $|$*amt*$|$ $\leq$ Φ.reserve(β), set β.fund := β.fund $+$ *amt*, β.allot := β.allot $+$ *amt* $+$ Φ.reserve(β).
2) Otherwise, set β.allot := β.allot $+$ Φ.reserve(β).

(D) Subroutine for channel closing

For each unbound Φ, if Φ.version $=$ β.biList(Φ.id) $+ 1$, or Φ.version $= 1$ and Φ.id $\notin$ β.biList, set β.balance(Φ.Irene) := β.balance(Φ.Irene) $+$ Φ.laShift(β). Refund β.balance(β.Alice) coins to β.Alice and β.balance(β.Bob) coins to β.Bob, send (closed) to β.users and close this contract.

Fig. 9: The contract functionality.

Besides, the timestamp is used to avoid the long time waiting for end users to confirm the request's effectiveness, especially when the user has replied with the signature on it but the request has not appeared on the blockchain. The timestamp indicates that the request is only valid within time $2+\Delta$, where 2 rounds are for the intermediate to collect signatures and Δ rounds are for submission. Receiving the request from the intermediate and confirmed by the environment, end users sign the bind request and send the signatures to the intermediate (Step (2), Figure 7(C)). They wait for at most $1+\Delta$ rounds to confirm whether the bind takes effect (Step (4), Figure 7(C)).

After the bind Φ is recorded on the blockchain, the intermediate can shift coins between the two bound channels. The process of coin shift is the same as described in Section III, including two phases of bind state update (Steps (1-4), Figure 8(D)) and channel state update (Steps (5-6), Figure 8(D)). Here we distinguish the second phase from the procedure (B). The former updates the bind version and intermediate's balance recorded in the channel state, while the latter is to perform payments in the channel. Besides different contents to be updated, the former is initiated by the intermediate of bound channels, and the latter could be initiated by any channel user. When the first phase completes, this coin shift takes effect. If it halts, users stop performing bind updates and initiates the unbinding. If the second phase halts, users stop conducting channel updates and initiate the channel closing.

The process of unbind is presented in Figure 8(E). Behaviors of the trigger and triggered contract are shown in Steps (3) and (4) of Figure 9 respectively. Any user could initiate the unbind (Step (1), Figure 8(E)) and the other users respond with the latest bind state (Step (2), Figure 8(E)). They would trigger the contract that timeout if it is not handled in a timely manner (Step (3), Figure 8(E)). From the aspect of the trigger contract, when triggered by one user to unbind Φ, the contract informs the other two users about the event and waits Δ for their responses (Step (3), Figure 9(B)). Finally, the contract decides the latest state, i.e., the valid state with the highest bind version, and triggers the other contract with the latest state. In addition, the contract sends or receives coins according to $\Phi.\mathsf{shift}$ recorded in the state, and withdraws the reserved amounts in the bind (Figure 9(C)).

Procedure (F) in Figure 8 shows the procedure of channel closing, corresponding to the contract procedures of Step (5) in Figure 9. Users submit the closing request and the latest channel state to the contract. If there are unbound channels, users proceed procedure (E) for each bind. The contract decides the latest channel state according to states submitted by users. Before refunding coins to users, the contract checks whether all binds have been dismissed. Besides, it needs to be verified that for each bind Φ, whether the version recorded in the channel state is consistent with that recorded in the contract (Figure 9(D)). If not, the contract recalculates the balance of $\Phi.\mathsf{Irene}$ according to $\Phi.\mathsf{laShift}$. Finally, the contract refunds coins to users and closes the channel.

Theorem 1: The protocol Shaduf running in the $\mathcal{C}$-hybrid world UC-realizes the ideal functionality $\mathcal{F}$ with respect to the global ledger $\mathcal{L}(\Delta)$.

Proof: The proof is in Appendix C. ■

VI. Implementation and Evaluation

To illustrate the feasibility of Shaduf, we provide a proof of concept implementation in Ethereum. We also conduct an experimental evaluation of Shaduf's performance in the payment channel network, to compare it with Revive, in a simulation of the Lighting Network. The source code is open on Github[2].

A. Implementation

Upon Ethereum, we implement the contract functionality shown in Figure 9 using the Solidity programming language. We focus on the on-chain costs running the Shaduf protocol, i.e., fees paid to miners when interacting with the contract. In Ethereum, it is calculated by the *gas*, which depends on the amount of data and the computation complexity of the contract call. We set the gas price as 20 Gwei (as of June 2021). Due to the recent Ether price fluctuations, we use the Nov 2020 exchange rate of 500 USD per Ether. Shaduf's costs are classified under each procedure and distinguished by the role one user plays, the initiator and the responder. The contract deployment costs 3.1M gas (31 USD). Since it can be mitigated by technical ways [8], we only consider the Shaduf running costs. Except for the channel state update and bind state update which are executed off-chain, costs of the other 4 procedures are displayed in Table II.

Costs of Shaduf. From Table II, channel opening costs the initiator 72k gas (0.72 USD) and the responder 57k gas (0.57 USD). Initiator of a bind, i.e., intermediate of two bound channels who submits the bind request to the blockchain, bears the total bind costs of 292k gas (2.92 USD). The unbinding could be initiated by any of its three users, and we measure the cost of unbinding in two cases. In the optimistic case, after the initiator submits the latest state to the contract, the other two users respond accordingly. In the pessimistic case, either the other two users do not respond in time thus the initiator needs to trigger the contract to finish the unbinding, or an expired state is submitted by the initiator thus the responder needs to submit the latest state. Similarly, we measure the costs of channel closing in both cases. The results show that the costs of Shaduf are minimized when all users behave honestly. The costliest procedure for both the initiator and the responder will be less than 520k gas (5.20 USD) even in the pessimistic case.

TABLE II: On-chain gas costs for executing Shaduf protocol, where "opti" represents the optimistic case and "pess" represents the pessimistic case.

	open	bind	unbind		close	
			opti	pess	opti	pess
Initiator	72342	292916	198069	277453	178108	206113
Responder	57934	0	112958	243299	62991	135396

Cost Comparison. Shaduf spends at most 813k gas (8.13 USD) for the bind and unbind procedures. Compared with Close-and-Reopen which costs 307k gas (3.07 USD) each time of channel refunding and LOOP which costs half the gas, costs of 3 times Close-and-Reopen and 6 times LOOP exceed Shaduf's costs, while Shaduf enables unlimited times of rebalancing after binding. Moreover, 3 times Close-and-Reopen and 6 times LOOP need at least 6Δ time, while Shaduf requires only Δ for the one-time binding.

B. Evaluation

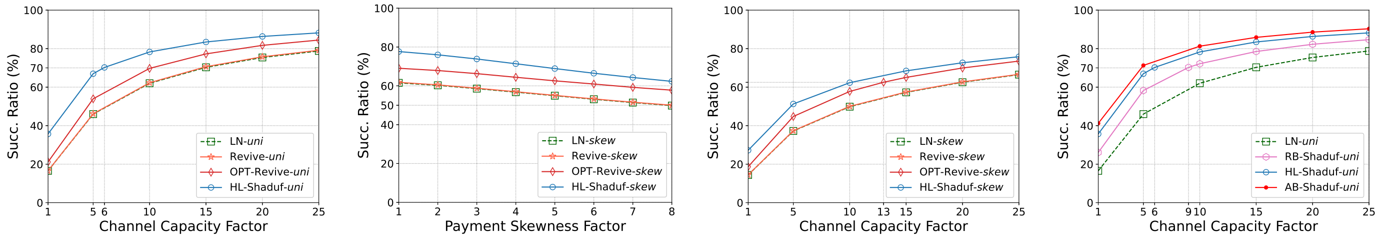
Generally, off-chain rebalancing methods have the potential to enhance the off-chain network's performance by enabling payments that may fail in the payment channel networks, such as the LN. Concretely, when performing a multi-hop payment in the LN, the sender selects the shortest path to the receiver from the network topology, and completes this payment with the help of the path users. However, the payment may fail due to any user, along the path, lacking coins in its hop. Revive and Shaduf alleviate this problem by refunding the depletion hop via coins in other channels, in global and local manners, respectively. To clarify their actual effects, we conduct the experimental evaluation and aim to answer the following questions,

- **Q1**: How does Shaduf perform in the off-chain network, in the respect of improving the network's success ratio and volume, and reducing users' deposits in channels, especially compared with LN and Revive?
- **Q2**: How does the binding strategy affect Shaduf's performance?

1) Evaluation setup: We setup the evaluation from two aspects, including the network topology and off-chain payments.

Network topology. Although Shaduf is constructed upon Turing-complete blockchain platforms, we use the real and the largest off-chain network, LN topology, to evaluate the performance. We get the snapshot of LN topology on 2021-03-31 (from https://ln.bigsun.xyz/), which contains 10,529 nodes and 38,910 channels. Since the initial balance allocation within each channel is invisible, we evenly assign the fund in each channel to two channel users to bootstrap the network.

[2] https://github.com/Lonely-Programmer/Shaduf



(a) Comparing succ. ratio of 4 schemes under uniform payment demands, with channel capacity factor from 1 to 25.

(b) Comparing succ. ratio of 4 schemes under skewed payment demands, with capacity factor 10 and payment skewness factor from 1 to 8.

(c) Comparing succ. ratio of 4 schemes under skewed payment demands, with skewness factor 8 and channel capacity factor from 1 to 25.

(d) Comparing succ. ratio of LN and 3 binding strategies of Shaduf under uniform payment demands, with channel capacity factor from 1 to 25.

Fig. 10: Effect of Shaduf varying the channel capacity, payment skewness and the binding strategy.

Off-chain payments simulation. Due to the fact that the off-chain payments (including the occurrence and amount) are invisible, to simulate the off-chain payments, we need to simulate both the sender-receiver pairs and the payment amounts between each pair.

- For choosing sender-receiver pairs, we consider two typical cases, uniform and skewed ones, separately. In the uniform case, a user performs payments with each other at the same probability. Therefore, senders and receivers are uniformly sampled from the network. In the skewed case, users such as customers pay more while merchants receive more. To simulate it, inspired by [25], we sample the sender and receiver from independent exponential distributions. The skewness of distribution is represented by the skewness factor, and a higher value indicates more skewed payments.
- As for off-chain transaction values, we simulate them by randomly sampling from the Bitcoin trace from 2021-03-01 to 2021-03-31. Furthermore, since LN is mainly used for high-frequency and low-amount transactions (or so-called micro-payments), to make the evaluation more compatible with the real off-chain network, we abandon the values higher than the median of the channel capacity. Finally, 2.65M transaction values are available.

2) Evaluation methodology: We design methods of evaluation for Revive and Shaduf. For the former, we implement the original Revive as suggested by [14], and optimize it to promote the performance. For the latter, we propose three different binding strategies.

Revive and its optimization. In Revive, the rebalancing demand is resolved in a global manner, i.e., the user completes the refund with other users who also have the refunding demand, under the help of a global fair leader. We simulate the rebalancing process in the following way. Firstly, each round of Revive starts by collecting the failed payments and regarding each user whose coins are insufficient for the payment to have the refunding demand, i.e., shifting coins in other channels to the current channel to continue the payment. Secondly, since more demands mean a higher possibility of directed cycles, to see the best performance of Revive, we assume that the fair leader starts this round of rebalancing when the number of demands reaches the maximum, i.e., 300 users or 1000 channels as suggested by [14]. After the generation of rebalancing transactions by the fair leader, we assume that all cycle users would cooperate with the rebalancing, after which all the failed payments are retried.

The performance of Revive can be improved by relaxing the refunding restrictions. A natural optimization of Revive could be achieved by exploiting the shortest cycles in the payment channel network. Instead of waiting for the other refunding demands to form directed cycles, a channel with rebalancing demand in the optimized Revive (OPT-Revive for short) gets help from users in its shortest cycles. In this way, the user is allowed to move coins in each other channel to the required channel, until sufficient coins are collected.

Shaduf. The refunding demand is resolved in a local manner, i.e., a user can shift coins within bound channels, with the cooperation of the other two users in each pair of bound channels. To simulate the case, we need to initiate the bindings for each user. For binding amounts, funds in a channel are evenly assigned to both end users to bind other channels, i.e., the maximum of funds that can be shifted out. When the channel is bound to multiple channels, the amount is equally distributed among them. As for binding strategy, a user with x channels can choose any subset from the C_x^2 pairs of bindings as its strategy. Three reasonable choices, including “high-to-low”, “random-bind” and “all-bind”, are described below.

In the first strategy, 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 on. In the second strategy, a user with x channels chooses bindings randomly, with the limitation that exactly $\lfloor 0.5 \cdot x \rfloor$ bindings are permitted. That is to say, the first and second strategies share the same number of bindings. In the third strategy, each pair of channels are bound. We denote the three strategies as “HL-Shaduf”, “RB-Shaduf” and “AB-Shaduf” respectively.

Ways of evaluation and comparison. Following the methodology and with the aforementioned evaluation setup, we carry out the evaluation. More concretely,

- To answer the first question (**Q1**), we test Shaduf’s performance with the “high-to-low” strategy, under the uniform (labeled with ‘-*uni*’) and skewed (labeled with ‘-*skew*’) payment demands separately. Performances of LN and (OPT-)Revive under the same setting are also displayed for comparison. To derive Shaduf’s effect on deposit reduction, we scale the channel capacity by a factor and see at which scale would (OPT-)Revive and Shaduf achieve the same performance as LN.
- To answer the second question (**Q2**), we vary Shaduf’s binding strategy and test the performance under the uni-

form payments.

It should be noted that in each evaluation, 50,000 sender-receiver pairs with corresponding payment values are chosen. Since randomness is introduced in the simulation of off-chain payments, we execute each evaluation 10 times and calculate the average. Both the success ratio and volume are taken as the metrics. Since the success volume is proportional to the success ratio, we put the volume data in Appendix D.

3) Evaluation results:

Success ratio in uniform payments. We vary the channel capacity from 1x to 25x and show the success ratio in uniform payments in Figure 10a. While Revive's performance is slightly better than LN, OPT-Revive achieves around 7% enhancement, and Shaduf achieves around 10% - 22% enhancement.

Success ratio in skewed payments. We test Shaduf's performance in skewed payments, under the fixed channel capacity factor 10. We increase the skewness factor until LN's success ratio drops to 50%, which we regard as the minimum-usable case. From the results displayed in Figure 10b, OPT-Revive achieves 7.5% enhancement and Shaduf achieves around doubled enhancement on average.

Deposit comparison in uniform payments. From Figure 10a, we see at which scale would OPT-Revive and Shaduf achieves the same success ratio of LN. For example, when the success ratio is 70%, the capacity factors of LN, OPT-Revive, and Shaduf are 15, 10, and 6, respectively. It means that OPT-Revive and Shaduf achieve the same performance with 0.67x and 0.4x capacity of LN. Correspondingly, they save 33% and 60% deposit respectively. On average, OPT-Revive and Shaduf saves 31% and 60% of the capacity, respectively.

Deposit comparison in skewed payments. Facing skewed payments, we consider the performances of OPT-Revive and Shaduf under the skewness factor 8. We vary the channel capacity and the results are displayed in Figure 10c. From the results, Shaduf improves the success ratio of LN and Revive by 12% on average. Correspondingly, 0.66x and 0.48x capacity is required for OPT-Revive and Shaduf to perform as effectively as LN, with 34% and 52% capacity saved. For example, when the success ratio is around 62%, the capacity factors of LN, OPT-Revive, and Shaduf are 20, 13, and 10, respectively.

Binding effect. Figure 10d shows the effect of the binding strategy. Shaduf under the "high-to-low" strategy performs better than the "random-bind" strategy, even under the same number of bindings, while the "all-bind" strategy performs best. In terms of the success ratio, the three strategies' enhancements vary from 6% to 28%. Under the same success ratio of LN, they achieve 37% - 75% deposit savings.

In summary, Shaduf's performance fluctuates under different payment demands and binding strategies. In general, Shaduf outperforms LN and Revive by 6% - 28% in success ratio, and reduces users' deposits by 37% - 75% under the same success ratio. A more dedicated binding strategy would further improve Shaduf's effect.

4) Discussion on the binding strategy: Apart from the three binding strategies described above, there are many other strategies for users to bind the channels. While more bindings cost higher on-chain fees, i.e., $2.92 times the number of bindings, in general, more payments would be enabled, from which the user would save the costs for on-chain channel refunding and charge fees for facilitating these payments. From another aspect, the deposit in channels could be reduced while maintaining the same performance. Therefore, users could trade off the costs and benefits of bindings, then decide and adjust the bindings according to the practical and dynamic demands.

VII. Privacy Enhanced Non-Cycle Rebalancing: Shaduf++

While Shaduf achieves secure off-chain coin shifts between bound channels, it does not consider the privacy issue, which is two-fold. On the one hand, the original channel privacy in the off-chain network should not be compromised. On the other hand, the introduced binding procedure should not bring potential privacy threats. As we analyzed below, the channel privacy is reserved while the binding procedure risks privacy leakage.

a) Channel Privacy: In the Lightning Network, the channel privacy means that the concrete fund allocation within one channel is invisible to anyone outside the channel. In other words, from their point of view, each channel user's balance is random in the range of 0 to the channel capacity. Recall that in Section III-B, in order to guarantee the balance security, the on-chain binding is required before performing off-chain coin shifts. Furthermore, to enable one channel to be bound to multiple channels, the upper limit of coin shifts is declared on the blockchain. Therefore, the relation of binding and the upper limit are public. Nevertheless, the privacy is not compromised in Shaduf. By dividing users outside the channel into two categories, i.e., blockchain observers and the user in its bound channel, we provide the non-formal privacy analysis from their perspectives, as shown below.

From the aspect of blockchain observers, no further information about the channel state is obtained since the coin shifts between bound channels are also off-chain performed among bound users. Moreover, the potential flowing-in coins from bound channels actually increase the channel capacity. Therefore, the range of each user's balance is expanded.

Compared with blockchain observers, the user in the bound channel knows the extra information, i.e., the concrete number of shifted coins between them. When coins are shifted out, the channel's capacity seems to be reduced, shrinking the channel users' balance range. However, the channel privacy is retained for two reasons. Firstly, the concrete balance allocation is still unknown. Secondly, since one channel could be bound to multiple channels, the user's balance range is affected by coins shifted from all bound channels.

In short, even though the binding relation and the upper limit of coin shift are public, Shaduf does not compromise the channel privacy.

b) Bind Privacy: The binding procedure risks the privacy leakage, more specifically, in two aspects,

- **Binding Relation.** For example, the fact that channels β and γ are bound is recorded on the blockchain and therefore being public.

- **Binding Amount.** For example, when channels β and γ are bound, the upper limits of coins that could be shifted from β to γ and in the opposite direction are also public.

From the above public information, users outside the channel could deduce some private information, such as the potential usage of the channel and the possible coin flow between two bound channels. For example, when most of the channel β's funds is bound to channel γ than other channels, it could be reasonably deduced that the intermediate plays the role of receiver in channel β and has higher demands of connecting β and γ.

To resist the potential leakage, we designed Shaduf++. Utilizing the cryptography tools, Shaduf++ hides the upper limit of coin shifts to users outside the channel with the commitment scheme, and enforces the correct protocol execution with the range proof. Based on that, the users could perform additional bindings with setting the upper limit to be 0. Since the non-channel users could not know the the upper limit, the bindings that actually take effect are hidden among all bindings, achieving the binding relation protection.

It should be noted that Shaduf++ is also built upon the Ethereum platform, which means that the balance allocation within one channel and coin shift between bound channels would be recorded on the blockchain in plaintext in cases of channel closing and unbinding. The content we aim to hide is the upper limit of coin shifts for each binding, i.e., $\Phi.\mathsf{reserve}$ for the binding Φ. At the same time, one channel's remaining binding amount, i.e., $\beta.\mathsf{allot}$ for the channel β, is updated along with each binding, which means it also needs to be hidden. Meanwhile, the correct protocol execution needs to be guaranteed. In summary, the privacy goal we aim to achieve while maintaining the correct protocol execution is as follows,

Binding Amount Hiding. For each channel, the upper limit of coin shifts for each binding and the remaining binding amount are only visible to the channel users.

Binding Amount Correctness. For each channel, one binding is valid when its upper limit of coin shift in this channel is no more than the channel's current remaining binding amount. Furthermore, the amount of coins shifted out of the channel is no more than the upper limit.

In the following, we describe procedures of Shaduf++, emphasizing on its differences compared with Shaduf, and give an informal analysis on its privacy and correctness.

A. *Procedures of Shaduf++*

On the basis of Shaduf, the 6 procedures in Shaduf++ are in the same message flow as Shaduf but with different message contents, which concern the privacy goal we aim to achieve. Moreover, among the 6 procedures, the `channel update` and `close` procedures have no relation to the privacy concern of binding and therefore stay unchanged.

Before the description on rest 4 procedures, we first introduce the notations we used. For the convenience of reading, we use the same notations as Shaduf, with the following exceptions,

Channel. For channel β, its remaining channel binding amount, which is denoted as $\beta.\mathsf{allot}$, is in the format of commitment. Specially, the value it commits and the opening are denoted as val_β and open_β, respectively.

Bind. For bind Φ, the upper limits of coin shifts in both directions are also in the format of commitment. Specially, for the upper limit of coins that could be shifted out of the bound channel γ, i.e., $\Phi.\mathsf{reserve}(\gamma)$, its committed value and the opening are denoted as $\mathsf{val}_{\Phi,\gamma}$ and $\mathsf{open}_{\Phi,\gamma}$, respectively.

Furthermore, we use the public parameter v_{max}, which specifies the maximum amount of coins that the underlying blockchain could handle. It is used in the range proofs that the values we aim to prove lie in the interval $[0, v_{max}]$.

Here we start the description on the 4 procedures and focus on the main changes related to the binding amounts. The detailed procedure description could be found in Figure 11 and Figure 12, which also show Shaduf for comparison.

a) Open: To open one channel with hidden remaining binding amount, the channel users initialize the amount using the commitment scheme, which could be verified by others. After the channel is opened, its remaining binding amount is updated along with each binding.

b) Bind: To bind two channels with hidden upper limits, the intermediate commits them using the commitment scheme and proves that the value is in the correct interval. Firstly, the value should be no less than 0. Secondly, the value should be no more than the channel's remaining binding amount, while both values are in the format of commitment.

The first proof could be generated using the range proof directly. The second proof equals that the difference of the two values is no less than 0. Utilizing the homomorphic property of the Pedersen commitment scheme, the division operation on the two commitments derives the commitment on the difference, whose range could be proved using the range proof.

It needs to be noted that the commitment of the binding's upper limit in one channel needs to be opened by both channel users. It is because without these information, the channel user could not generate the range proof for further bindings, which means one channel only allows one user to bind, limiting the application.

After the bind is confirmed, the channel's remaining binding amount is updated to the commitment of the difference. The channel users update its plaintext value and the opening locally.

c) Bind Update: When performing the bind update, the intermediate needs to prove that the amount of total coins shifted out of one channel after this update is no more than the upper limit, while the latter is in the format of commitment. Committing the amount of shifted coins using the randomness 0, the proof could also be generated utilizing the homomorphic property of the Pedersen commitment scheme and range proof.

d) Unbind: In the case of unbinding, the user needs to submit the latest bind state to the contract along with a proof that the coins shifted out of one channel are no more than the upper limit, which is in the format of commitment. It can be seen that the required proof equals the one attached when performing the bind update. Therefore users could store

- Assume the following messages concerning the channel β, which is composed by Lara and Irene (plays the role of $\beta.\mathsf{Alice}$).
- Assume the following messages concerning the bind Φ, which binds channels β and γ and takes Irene as the intermediate.

Lara		**Irene**
	(A) Open	
		Input: the channel β 1. Generate the opening request. • **Shaduf.** $\perp$ • **Shaduf++.** Set $\beta.\mathsf{allot} := \mathsf{Commit}(\beta.\mathsf{fund}; 0)$ 2. Send the opening request to the contract, which checks the correctness as follows. • **Shaduf.** Check $\beta.\mathsf{allot} = \beta.\mathsf{fund}$. • **Shaduf++.** Check $\mathsf{Open}(\beta.\mathsf{allot}, \beta.\mathsf{fund}, 0)$
3. Update the local information when β is opened. • **Shaduf.** $\perp$ • **Shaduf++.** Set $\mathsf{val}_\beta := \beta.\mathsf{fund}$, $\mathsf{open}_\beta := 0$		3. Update the local information when β is opened. • **Shaduf.** $\perp$ • **Shaduf++.** Set $\mathsf{val}_\beta := \beta.\mathsf{fund}$, $\mathsf{open}_\beta := 0$
	(C) Bind	
		Input: the bind Φ 1. Generate the binding request. • **Shaduf.** Set $msg_\eta := \Phi$ for $\eta \in \{\beta, \gamma\}$ • **Shaduf++.** For $\eta \in \{\beta, \gamma\}$: - Set $\mathsf{val}_{\Phi,\eta} := \Phi.\mathsf{reserve}(\eta)$ - $\mathsf{c}_{\Phi,\eta} \leftarrow \mathsf{Commit}(\mathsf{val}_{\Phi,\eta}; \mathsf{open}_{\Phi,\eta})$ - Set $\Phi.\mathsf{reserve}(\eta) := \mathsf{c}_{\Phi,\eta}$ - $\pi_{\Phi,\eta} \leftarrow \mathcal{P}(\mathsf{c}_{\Phi,\eta}, \mathsf{val}_{\Phi,\eta}, \mathsf{open}_{\Phi,\eta})$ - $\pi'_{\Phi,\eta} \leftarrow \mathcal{P}(\eta.\mathsf{allot}/\mathsf{c}_{\Phi,\eta}, \mathsf{val}_\eta - \mathsf{val}_{\Phi,\eta}, \mathsf{open}_\eta - \mathsf{open}_{\Phi,\eta})$ - Set $msg_\eta := \{\Phi, \mathsf{val}_{\Phi,\eta}, \mathsf{open}_{\Phi,\eta}, \pi_{\Phi,\neg\eta}, \pi'_{\Phi,\neg\eta}\}$ 2. Send the binding request to the end users.
	$\xleftarrow{msg_\beta}$	
3. Verify the correctness of the binding request. • **Shaduf.** - Check $\Phi.\mathsf{reserve}(\eta) \leq \eta.\mathsf{allot}$ for $\eta \in \{\beta, \gamma\}$ • **Shaduf++.** - Check $0 \leq \mathsf{val}_{\Phi,\beta} \leq \mathsf{val}_\beta$ - Check $\mathsf{Open}(\Phi.\mathsf{reserve}(\beta), \mathsf{val}_{\Phi,\beta}, \mathsf{open}_{\Phi,\beta})$ - Check $\mathcal{V}(\Phi.\mathsf{reserve}(\gamma), \pi'_{\Phi,\gamma})$ - Check $\mathcal{V}(\gamma.\mathsf{allot}/\Phi.\mathsf{reserve}(\gamma), \pi_{\Phi,\gamma})$ Proceed if the checks pass and abort otherwise.		
		4. Send (Φ, $\pi_{\Phi,\beta}$,$\pi'_{\Phi,\beta}$,$\pi_{\Phi,\gamma}$,$\pi'_{\Phi,\gamma}$) to the blockchain and for $\eta \in \{\beta, \gamma\}$, $\mathcal{C}(\eta)$ does the following checks. • **Shaduf.** Check $\Phi.\mathsf{reserve}(\eta) \leq \eta.\mathsf{allot}$ • **Shaduf++.** - Check $\mathcal{V}(\Phi.\mathsf{reserve}(\eta), \pi_{\Phi,\gamma})$ - Check $\mathcal{V}(\eta.\mathsf{allot}/\Phi.\mathsf{reserve}(\eta), \pi'_{\Phi,\eta})$
5. Update the local information when Φ is bound. • **Shaduf.** - Set $\beta.\mathsf{allot} := \beta.\mathsf{allot} - \Phi.\mathsf{reserve}(\beta)$ • **Shaduf++.** - Store $\mathsf{val}_{\Phi,\beta}$, $\mathsf{open}_{\Phi,\beta}$ - Set $\beta.\mathsf{allot} := \beta.\mathsf{allot}/\Phi.\mathsf{reserve}(\beta)$ - Set $\mathsf{val}_\beta := \mathsf{val}_\beta - \mathsf{val}_{\Phi,\beta}$ - Set $\mathsf{open}_\beta := \mathsf{open}_\beta - \mathsf{open}_{\Phi,\beta}$		5. Update the local information when Φ is bound. • **Shaduf.** For $\eta \in \{\beta, \gamma\}$: - Set $\eta.\mathsf{allot} := \eta.\mathsf{allot} - \Phi.\mathsf{reserve}(\eta)$ • **Shaduf++.** For $\eta \in \{\beta, \gamma\}$: - Store $\mathsf{val}_{\Phi,\eta}$, $\mathsf{open}_{\Phi,\eta}$ - Set $\eta.\mathsf{allot} := \eta.\mathsf{allot}/\mathsf{c}_{\Phi,\eta}$ - Set $\mathsf{val}_\eta := \mathsf{val}_\eta - \mathsf{val}_{\Phi,\eta}$ - Set $\mathsf{open}_\eta := \mathsf{open}_\eta - \mathsf{open}_{\Phi,\eta}$

Fig. 11: Handling of binding amount related messages in procedures (A) Open and (C) Bind of Shaduf and Shaduf++.

- Assume the following messages concerning the channel β, which is composed by Lara and Irene.
- Assume the following messages concerning the bind Φ, which binds channels β and γ and takes Irene as the intermediate.

Lara | **Irene**

(D) Bind Update

Irene

Input: the coin shift $\widetilde{\theta}$

1. Generate the new bind state.

- **Shaduf.** Set $\Phi.\mathsf{shift} := \Phi.\mathsf{shift} + \widetilde{\theta}$, $msg := \perp$
- **Shaduf++.**
  - Denote $amt := \Phi.\mathsf{shift}(\eta) \leq 0$ that $\eta \in \{\beta, \gamma\}$
  - $\mathsf{c} \leftarrow \mathsf{Commit}(\Phi.\mathsf{shift}(\eta); 0)$
  - $\pi \leftarrow \mathcal{P}(\Phi.\mathsf{reserve}(\eta) \cdot \mathsf{c}, \mathsf{val}_{\Phi,\eta} + amt, \mathsf{open}_{\Phi,\eta})$
  - Set $\Phi.\mathsf{shift} := \Phi.\mathsf{shift} + \widetilde{\theta}$, $msg := \pi$

2. Send the update request to the end users.

$$\xleftarrow{\widetilde{\theta},\ \Phi.\mathsf{shift},\ msg}$$

Lara

3. Verify the correctness of received coin shift.

Check $\beta.\mathsf{balance}(\mathsf{Irene}) \geq |\widetilde{\theta}(\beta)|$ if $\widetilde{\theta}(\beta) \leq 0$

Denote $amt := \Phi.\mathsf{shift}(\eta) \leq 0$ that $\eta \in \{\beta, \gamma\}$

- **Shaduf.**
  - Check $\Phi.\mathsf{reserve}(\eta) \geq |\Phi.\mathsf{shift}(\eta)|$
- **Shaduf++.**
  - $\mathsf{c} \leftarrow \mathsf{Commit}(\Phi.\mathsf{shift}(\eta); 0)$
  - Check $\mathcal{V}(\Phi.\mathsf{reserve}(\eta) \cdot \mathsf{c}, \pi)$

Proceed if the checks pass and abort otherwise.

4. Update local information when $\widetilde{\theta}$ is confirmed.

Set $\beta.\mathsf{balance}(\mathsf{Irene}) := \beta.\mathsf{balance}(\mathsf{Irene}) + \widetilde{\theta}(\beta)$

- **Shaduf.** Update Φ's state.
- **Shaduf++.** Update Φ's state and its proof π.

Irene

4. Update local information when $\widetilde{\theta}$ is confirmed.

Set $\eta.\mathsf{balance}(\mathsf{Irene}) := \eta.\mathsf{balance}(\mathsf{Irene}) + \widetilde{\theta}(\eta)$ for $\eta \in \{\beta, \gamma\}$

- **Shaduf.** Update Φ's state.
- **Shaduf++.** Update Φ's state and its proof π.

(E) Unbind

Lara

1. Submit the information to the contract, which checks the correctness as the blue parts show.

- **Shaduf.** Submit Φ's latest state.
- **Shaduf++.** Submit Φ's latest state and π.

2. Update the local information when Φ is unbound.

- **Shaduf.**
  - Set $\beta.\mathsf{allot} := \beta.\mathsf{allot} + \Phi.\mathsf{reserve}(\beta) + \Phi.\mathsf{shift}(\beta)$
- **Shaduf++.**
  - Set $\beta.\mathsf{allot} := \beta.\mathsf{allot} \cdot \Phi.\mathsf{reserve}(\beta) \cdot \mathsf{Commit}(\Phi.\mathsf{shift}(\beta); 0)$
  - Set $\mathsf{val}_\beta := \mathsf{val}_\beta + \mathsf{val}_{\Phi,\beta} + \Phi.\mathsf{shift}(\beta)$
  - Set $\mathsf{open}_\beta := \mathsf{open}_\beta + \mathsf{open}_{\Phi,\beta}$

Irene

1. Submit the information to the contract, which checks the correctness as the blue parts show.

- **Shaduf.** Submit Φ's latest state.
- **Shaduf++.** Submit Φ's latest state and π.

2. Update the local information when Φ is unbound.

- **Shaduf.** For $\eta \in \{\beta, \gamma\}$,
  - Set $\eta.\mathsf{allot} := \eta.\mathsf{allot} + \Phi.\mathsf{reserve}(\eta) + \Phi.\mathsf{shift}(\eta)$
- **Shaduf++.** For $\eta \in \{\beta, \gamma\}$,
  - Set $\eta.\mathsf{allot} := \eta.\mathsf{allot} \cdot \Phi.\mathsf{reserve}(\eta) \cdot \mathsf{Commit}(\Phi.\mathsf{shift}(\eta); 0)$
  - Set $\mathsf{val}_\eta := \mathsf{val}_\eta + \mathsf{val}_{\Phi,\eta} + \Phi.\mathsf{shift}(\eta)$
  - Set $\mathsf{open}_\eta := \mathsf{open}_\eta + \mathsf{open}_{\Phi,\eta}$

Fig. 12: Handling of binding amount related messages in procedures (D) Bind Update and (E) Unbind of Shaduf and Shaduf++.

it when receiving from the intermediate and submit it in case of unbinding.

After the unbinding, the channel's remaining binding amount is updated accordingly. The channel users also update the plaintext value and the opening locally.

B. Analysis of Shaduf++

Below we provide an informal analysis on the privacy and correctness of Shaduf++. A formal proof would be provided in the future work.

a) Binding Amount Hiding: For users outside one channel, their views on the channel's remaining binding amount are a series of updated commitments and range proofs that the remained binding amount is no less than 0 after confirming each binding. Similarly, their views on each binding's upper limit of coin shifts include its commitment, range proofs that the value is no less than 0 and the coins shifted out of the channel are no more than the upper limit. Especially, for the user in its bound channel, the view consists of more ranges proofs that after each coin shift, the resulted shifted coins are no more than the upper limit.

Since the committed value could not be derived by the commitment, the plaintext remained binding amount and the upper limit of coin shifts could not be known by the non-channel users. Moreover, due to the zero-knowledge property of the range proof scheme, the proofs do not leak any information about the value except that the value is in the correct range. Therefore, the bind amount hiding is achieved.

Furthermore, since one commitment could not be opened to another value, both channel users would know the value and opening of channel's remaining binding amount, which allows both users to generate range proofs for further bindings.

b) Binding Amount Correctness: Due to the completeness property of the range proof, the bindings whose upper limit is no less than 0 and no more than the channel's remaining binding amount could pass the verification and take effect. Similarly, proof on the binding state whose coins shifted out one channel are no more than the upper limit could always be accepted and therefore the state proved valid.

Due to the soundness property of the range proof, the bindings whose upper limit is not in the correct range would not be accepted. It holds the same for the verification that the amount of shifted coins is no more than the upper limit.

VIII. Related work

The issue of channel fund depletion is first proposed by Revive [14]. With the atomic multi-channel update protocol [10], Revive is Bitcoin-compatible and can be deployed to, for example, LN. Spider [25] solves the problem in a different way, which proposes a balanced routing scheme to avoid fund depletion in channels facing skewed payment flows. Similarly, to avoid payment failure due to depleted funds in channels, Li *et al.* [15] plan the deposition in channels to satisfy the payment demands. Both approaches aim to avoid fund depletion, which are orthogonal to Revive and our work since we provide solutions for channels that have already been depleted. The two kinds of solutions can be combined to achieve a better performance in maintaining balanced channels.

1) Payment channel: Many variants of the payment channel have been extensively studied. Bolt [11] constructs anonymous channels where payments in one channel are unlinkable. Payment channels are built upon multiple blockchain platforms [21], [28], and extended to state channels where arbitrary state transitions are enabled [1], [7], [9], [20].

2) Payment channel network: The routing strategies have been studied in [16], [24], [25], [27]. Tumblebit [13] constructs payment unlinkable PCH. Maintaining the privacy feature, A^2L [26] enhances Tumblebit's efficiency and interoperability. Malavolta *et al.* [18] report the wormhole attack and propose the secure and privacy-preserving PCN construction. Sprites [20] reduces the collateral costs of multi-hop payments. Egger *et al.* [10] achieve the collateral reduction on Bitcoin-compatible platforms. Aumayr *et al.* [3] build the secure and collateral-reduced multi-hop payments in a single round. Perun [8] proposes the concept of the virtual channel, via which users with one-hop distance can perform off-chain payments directly. The virtual channel is also enabled on Bitcoin-compatible platforms [2]. Dziembowski *et al.* [9] connect arbitrary users directly via virtual channels and build the state channel network. The virtual channel is also extended to multi-party where multiple parties can perform state transitions [7]. We refer readers to [12] for a comprehensive understanding of the payment channel and the network.

IX. Conclusion

In this paper, we have proposed a non-cycle off-chain rebalancing scheme named Shaduf. It allows users to rebalance depleted channels via exploiting coins from other channels in more general cases. We formalize the protocol in the UC framework and prove its security. Experimental evaluation shows that Shaduf outperforms LN and Revive in raising the performance of the off-chain payment network. We also remark that Shaduf is an orthogonal and compatible work to the well-studied off-chain routing methods, therefore we believe the joint applications of Shaduf with them will perform even better. Moreover, the privacy-enhanced Shaduf, named Shaduf++, protects both the balance within a single channel and the bind privacy among channels as well.

References

[1] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized channels from limited blockchain scripts and adaptor signatures," in *Advances in Cryptology - ASIACRYPT*, vol. 13091. Springer, 2021, pp. 635–664.

[2] L. Aumayr, M. Maffei, O. Ersoy, A. Erwig, S. Faust, S. Riahi, K. Hostáková, and P. Moreno-Sanchez, "Bitcoin-compatible virtual channels," in *42nd IEEE Symposium on Security and Privacy, SP*. IEEE, 2021, pp. 901–918.

[3] L. Aumayr, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Blitz: Secure multi-hop payments without two-phase commits," in *30th USENIX Security Symposium*. USENIX Association, 2021, pp. 4043–4060.

[4] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *IEEE Symposium on Security and Privacy, SP*. IEEE Computer Society, 2018, pp. 315–334.

[5] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd Annual Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 2001, pp. 136–145.

[6] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC*, vol. 4392. Springer, 2007, pp. 61–85.

[7] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, vol. 11476. Springer, 2019, pp. 625–656.

[8] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *IEEE Symposium on Security and Privacy, SP*. IEEE, 2019, pp. 106–123.

[9] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 2018, pp. 949–966.

[10] C. Egger, P. Moreno-Sanchez, and M. 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, CCS*. ACM, 2019, pp. 801–815.

[11] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 2017, pp. 473–489.

[12] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *Financial Cryptography and Data Security - 24th International Conference, FC*, ser. Lecture Notes in Computer Science, vol. 12059. Springer, 2020, pp. 201–226.

[13] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub," in *24th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2017.

[14] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 2017, pp. 439–453.

[15] P. Li, T. Miyazaki, and W. Zhou, "Secure balance planning of off-blockchain payment channel networks," in *39th IEEE Conference on Computer Communications, INFOCOM*. IEEE, 2020, pp. 1728–1737.

[16] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," in *24th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2017.

[17] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM, 2017, pp. 455–471.

[18] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *26th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2019.

[19] V. Mavroudis, K. Wüst, A. Dhar, K. Kostiainen, and S. Capkun, "Snappy: Fast on-chain payments with practical collaterals," in *27th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2020.

[20] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Financial Cryptography and Data Security - 23rd International Conference, FC*, vol. 11598. Springer, 2019, pp. 508–526.

[21] P. Moreno-Sanchez, A. Blue, D. V. Le, S. Noether, B. Goodell, and A. Kate, "DLSAG: non-interactive refund transactions for interoperable payment channels in monero," in *Financial Cryptography and Data Security - 24th International Conference, FC*, vol. 12059. Springer, 2020, pp. 325–345.

[22] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference*, vol. 576. Springer, 1991, pp. 129–140.

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

[24] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *25th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2018.

[25] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. C. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI*. USENIX Association, 2020, pp. 777–796.

[26] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A^{2}l: Anonymous atomic locks for scalability in payment channel hubs," in *42nd IEEE Symposium on Security and Privacy, SP*. IEEE, 2021, pp. 1834–1851.

[27] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: efficient dynamic routing for offchain networks," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies, CoNEXT*. ACM, 2019, pp. 370–381.

[28] Y. Zhang, Y. Long, Z. Liu, Z. Liu, and D. Gu, "Z-channel: Scalable and efficient scheme in zerocash," in *Information Security and Privacy - 23rd Australasian Conference, ACISP*, vol. 10946. Springer, 2018, pp. 687–705.

Appendix A
Simplifying the Ideal functionality description

The checks that the ideal functionality $\mathcal{F}$ would do on received messages are shown in Figure 13.

Appendix B
Simplifying the Shaduf Protocol description

The checks that an honest user in the Shaduf protocol would do on the received messages are shown in Figure 14.

Appendix C
Proof of Theorem 1

To achieve the indistinguishability between the ideal world and the real world, the simulator $\mathcal{S}$ sends inputs to the ideal functionality $\mathcal{F}$ observing behaviors of the corrupted parties and sends messages to the corrupted parties on behalf of the contract functionality $\mathcal{C}$ and the honest parties.

At the beginning, $\mathcal{S}$ corrupts the parties that $\mathcal{A}$ corrupts. $\mathcal{S}$ also generates the public-private key pairs for all parties and passes all public keys to the corrupted parties. For each corrupted party, its private key is sent along. Simulation on each procedure is shown in Figure 15 and 16. In each procedure, we ignore cases where involved parties are all honest or corrupted, since $\mathcal{S}$ simulates behaviors of the corrupted parties and we aim to protect the balance security of the honest parties. Moreover, $\mathcal{S}$ checks the correctness of messages received from the corrupted parties and updates the local storage, as the honest users would do. For the sake of presentation, the checks and updates are omitted here. We refer to Figure 14 for the checks and Figure 7 & 8 for the updates.

Appendix D
Success volume of evaluation

The total payment volumes in the evaluation is 94 BTC. The data of success volume displayed in Figure 17a, 17b, 17c, and 17d correspond to the four evaluation that is shown in Figure 10a, 10b, 10c, and 10d, respectively.

(A) **Open**

Upon (`open`, β) $\overset{t}{\hookleftarrow}$ $\beta.\mathsf{Alice}$, do the following checks. (1) $\Gamma(\beta.\mathsf{id}) = \perp$, (2) there is no channel γ with $\gamma.\mathsf{id} = \beta.\mathsf{id}$ currently being opened, (3) β is in the correct channel format, and (4) both $\beta.\mathsf{Alice}$ and $\beta.\mathsf{Bob}$ have sufficient coins on $\mathcal{L}$ for the channel deposit. If any of the above checks fails, ignore the message. Otherwise, proceed as the functionality $\mathcal{F}$.

(B) **Channel Update**

Upon (`chan-update`, id, θ) $\overset{t}{\hookleftarrow}$ P, do the following checks. (1) $\beta := \Gamma(id) \neq \perp$, (2) $P \in \beta.\mathsf{users}$, (3) there is no other channel update being performed, (4) θ is in the correct payment format, and (5) $\beta.\mathsf{balance}(Q) + \theta(Q) \geq 0$ for $Q \in \beta.\mathsf{users}$. If any of the above checks fails, ignore the message. Otherwise, proceed as the functionality $\mathcal{F}$.

(C) **Bind**

Upon (`bind`, Φ) $\overset{t}{\hookleftarrow}$ I, do the following checks. (1) $\Lambda(\Phi.\mathsf{id}) = \perp$, (2) $I = \Phi.\mathsf{Irene}$, (3) there is no bind Φ' with $\Phi.\mathsf{id} = \Phi'.\mathsf{id}$ currently being bound, (4) Φ is in the correct bind format, and (5) $\gamma.\mathsf{allot} \geq \Phi.\mathsf{reserve}(\gamma)$ for $\gamma \in \Phi.\mathsf{channels}$. If any of the above checks fails, ignore the message. Otherwise, proceed as the functionality $\mathcal{F}$.

(D) **Bind Update**

Upon (`bind-update`, $\widetilde{id}, \tilde{\theta}$) $\overset{t}{\hookleftarrow}$ I, do the following checks. (1) $\Phi := \Lambda(\widetilde{id}) \neq \perp$, (2) $I = \Phi.\mathsf{Irene}$, (3) there is no other bind update being performed, (4) $\tilde{\theta}$ is in the correct coin shift format, (5) let $\tilde{\theta}(\gamma) \leq 0$ that $\gamma \in \Phi.\mathsf{channels}$, then $|\tilde{\theta}(\gamma)| \leq \gamma.\mathsf{balance}(I)$, and (6) let $\Phi.\mathsf{shift}(\gamma) + \tilde{\theta}(\gamma) \leq 0$ that $\beta \in \Phi.\mathsf{channels}$, then $|\Phi.\mathsf{shift}(\beta) + \tilde{\theta}(\beta)| \leq \Phi.\mathsf{reserve}(\beta)$. If any of the above checks fails, ignore the message. Otherwise, proceed as the functionality $\mathcal{F}$.

(E) **Unbind**

Upon (`unbind`, $\widetilde{id}$) $\overset{t}{\hookleftarrow}$ P, do the following checks. (1) $\Phi := \Lambda(\widetilde{id}) \neq \perp$, and (2) $P \in \Phi.\mathsf{users}$. If any of the above checks fails, ignore the message. Otherwise, proceed as the functionality $\mathcal{F}$.

(F) **Close**

Upon (`close`, id) $\overset{t}{\hookleftarrow}$ P, do the following checks. (1) $\beta := \Gamma(id) \neq \perp$, and (2) $P \in \beta.\mathsf{users}$. If any of the above checks fails, ignore the message. Otherwise, proceed as the functionality $\mathcal{F}$.

Fig. 13: Checks on messages received by the ideal functionality $\mathcal{F}$

(A) **Open**

Upon (`open`, β) $\overset{t}{\hookleftarrow} \mathcal{Z}$, $\beta.\mathsf{Alice}$ does the following checks. (1) $\Gamma^A(\beta.\mathsf{id}) =\perp$, (2) there is no channel γ with $\gamma.\mathsf{id} = \beta.\mathsf{id}$ currently being opened, (3) β is in the correct channel format, and (4) both $\beta.\mathsf{Alice}$ and $\beta.\mathsf{Bob}$ have sufficient coins in $\mathcal{L}$ for the channel deposit. If any of the above checks fails, ignore the message. Otherwise, proceed as the protocol.

(B) **Channel Update**

Upon (`chan-update`, id, θ) $\overset{t}{\hookleftarrow} \mathcal{Z}$, P does the following checks. (1) $\beta := \Gamma^P(id) \neq\perp$, (2) there is no other channel update being performed, (3) θ is in the correct payment format, and (4) $\beta.\mathsf{balance}(Q) + \theta(Q) \geq 0$ for $Q \in \beta.\mathsf{users}$. If any of the above checks fails, ignore the message. Otherwise, proceed as the protocol.

Upon (`chan-update`, $id, msg = (ver, \alpha, biList)$, σ_P) $\overset{\tau}{\hookleftarrow} P$, Q does the following checks. (1) $\beta := \Gamma^Q(id) \neq\perp$, (2) there is no other channel update being performed, (3) $P = \beta.\mathsf{other\text{-}user}(Q)$, (4) $ver = \beta^Q.\mathsf{version} + 1$, (5) $\theta := \alpha - \beta^Q.\mathsf{balance}$ is in the correct payment format, (6) $\alpha(P) \geq 0$ for $P \in \beta.\mathsf{users}$, (7) $biList = \beta^Q.\mathsf{biList}$, and (8) σ_P is P's signature on msg. If any of the above checks fails, ignore the message. Otherwise, proceed as the protocol.

(C) **Bind**

Upon (`bind`, Φ) $\overset{t}{\hookleftarrow} \mathcal{Z}$, I does the following checks. (1) $\Lambda^I(\Phi.\mathsf{id}) =\perp$, (2) there is no bind Φ' with $\Phi.\mathsf{id} = \Phi'.\mathsf{id}$ currently being bound, (3) Φ is in the correct bind format, and (4) $\gamma.\mathsf{allot}(\gamma) \geq \Phi.\mathsf{reserve}(\gamma)$ for $\gamma \in \Phi.\mathsf{channels}$. If any of the above checks fails, ignore the message. Otherwise, proceed as the protocol.

Upon (`bind`, $\Phi, \tau - 1, \sigma_I$) $\overset{\tau}{\hookleftarrow} I$, P does the following checks. (1) $\Lambda^P(\Phi.\mathsf{id}) =\perp$, (2) $I = \Phi.\mathsf{Irene}$, (3) $P \in \Phi.\mathsf{users}$, (4) there is no bind Φ' with $\Phi.\mathsf{id} = \Phi'.\mathsf{id}$ currently being bound, (5) Φ is in the correct bind format, (6) $\gamma.\mathsf{allot} \geq \Phi.\mathsf{reserve}(\gamma)$ for $\gamma \in \Phi.\mathsf{channels}$, and (7) σ_I is I's signature on $msg = (\Phi, \tau - 1)$. If any of the above checks fails, ignore the message. Otherwise, proceed as the protocol.

(D) **Bind Update**

Upon (`bind-update`, $\widetilde{id}$, $\tilde{\theta}$) $\overset{t}{\hookleftarrow} \mathcal{Z}$, I does the following checks. (1) $\Phi := \Lambda^I(\widetilde{id}) \neq\perp$, (2) $I = \Phi.\mathsf{Irene}$, (3) there is no other bind update being performed, (4) $\tilde{\theta}$ is in the correct coin shift format, (5) let $\tilde{\theta}(\gamma) \leq 0$ that $\gamma \in \Phi.\mathsf{channels}$, then $|\tilde{\theta}(\gamma)| \leq \gamma.\mathsf{balance}(I)$, and (6) let $\Phi^I.\mathsf{shift}(\beta) + \tilde{\theta}(\beta) \leq 0$ that $\beta \in \Phi.\mathsf{channels}$, then $|\Phi^I.\mathsf{shift}(\beta) + \tilde{\theta}(\beta)| \leq \Phi.\mathsf{reserve}(\beta)$. If any of the above checks fails, ignore the message. Otherwise, proceed as the protocol.

Upon (`bind-update`, $\widetilde{id}, msg = (ver, shift, \tilde{\theta}), \sigma_I$) $\overset{\tau}{\hookleftarrow} I$, P do the following checks. (1) $\Phi := \Lambda^P(\widetilde{id}) \neq\perp$, (2) $I = \Phi.\mathsf{Irene}$, (3) there is no other bind update being performed, (4) $ver = \Phi^P.\mathsf{version} + 1$, (5) $shift = \Phi^P.\mathsf{shift} + \tilde{\theta}$, (6) $\tilde{\theta}$ is in the correct coin shift format, (7) let $\gamma \in \Phi.\mathsf{channels}$ be the channel where P is in, if $\tilde{\theta}(\gamma) \leq 0$, then $\gamma^P.\mathsf{balance}(I) \geq \tilde{\theta}(\gamma)|$, (8) if $shift(\gamma) \leq 0$, then $\Phi^P.\mathsf{reserve}(\gamma) \geq |shift(\gamma)|$. Otherwise, $\Phi.\mathsf{reserve}(\beta) \geq |shift(\beta)|$ and (9) σ_I is I's signature on msg. If any of the above checks fails, ignore the message. Otherwise, proceed as the protocol.

(E) **Unbind**

Upon (`unbind`, $\widetilde{id}$) $\overset{t}{\hookleftarrow} \mathcal{Z}$, P checks that $\Phi := \Lambda^P(\widetilde{id}) \neq\perp$.

(F) **Close**

Upon (`close`, id) $\overset{t}{\hookleftarrow} \mathcal{Z}$, P checks $\beta := \Gamma^P(id) \neq\perp$.

Fig. 14: Checks on messages received by the honest user in the Shaduf protocol.

We abbreviate $A := \beta.\mathsf{Alice}$, $B := \beta.\mathsf{Bob}$, $I := \Phi.\mathsf{Irene}$. Furthermore, we denote $Q := \beta.\mathsf{other\text{-}user}(P)$ for $P \in \beta.\mathsf{users}$.

(A) Open

Case A is honest and B is corrupt

Upon A sending (`open`, β) $\overset{t}{\hookrightarrow} \mathcal{F}$, send (`open`, β) $\overset{t}{\hookrightarrow} \mathcal{C}(\beta)$ on behalf of A. Assume it reaches $\mathcal{L}$ in round $\tau \leq t + \Delta$. If B sends (`open`) $\overset{\tau}{\hookrightarrow} \mathcal{C}(\beta)$, send (`open`) $\overset{\tau}{\hookrightarrow} \mathcal{F}$ on behalf of B. Otherwise, upon A sending (`refund`, β) to $\mathcal{F}$, send (`timeout`) to $\mathcal{C}(\beta)$ on behalf of A.

Case A is corrupt and B is honest

Upon A sending (`open`, β) $\overset{t}{\hookrightarrow} \mathcal{C}(\beta)$, send (`open`, β) $\overset{t}{\hookrightarrow} \mathcal{F}$ on behalf of A. Assume it reaches $\mathcal{L}$ in round $\tau \leq t + \Delta$. If B sends (`open`) $\overset{\tau}{\hookrightarrow} \mathcal{F}$, send (`open`) $\overset{\tau}{\hookrightarrow} \mathcal{C}(\beta)$ on behalf of B. Otherwise, upon B sending (`timeout`) to $\mathcal{C}(\beta)$, send (`refund`, β) to $\mathcal{F}$ on behalf of B.

(B) Channel Update

Case P is honest and Q is corrupt

Upon P sending (`chan-update`, id, θ) $\overset{t}{\hookrightarrow} \mathcal{F}$, let $\beta := \Gamma^P(id)$, generate signature σ_P on $msg = (\beta^P.\mathsf{version} + 1, \beta^P.\mathsf{balance} + \theta, \beta^P.\mathsf{biList})$, and (`chan-update`, id, msg, σ_P) $\overset{t}{\hookrightarrow} Q$ on behalf of P. If Q sends his signature on msg to P in round $t + 1$, send (`chan-update-ok`) to $\mathcal{F}$ on behalf of Q. Otherwise, initiate channel closing on behalf of P.

Case P is corrupt and Q is honest

Upon P sending (`chan-update`, $id, msg = (ver, \alpha, biList)$, σ_P) $\overset{t}{\hookrightarrow} Q$, let $\beta := \Gamma^Q(id)$, send (`chan-update`, id, $\alpha - \beta^Q.\mathsf{balance}$) $\overset{t}{\hookrightarrow} \mathcal{F}$ on behalf of P. Else, stop the procedure. If Q sends (`chan-update-ok`) $\xhookrightarrow{t+1} \mathcal{F}$, generate Q's signature σ_Q on msg and send it to P on behalf of Q.

(C) Bind

Case I is honest

Upon I sending (`bind`, Φ) $\overset{t}{\hookrightarrow} \mathcal{F}$, generate I's signature σ_I on (Φ, t), send (`bind`, Φ, t, σ_I) to $\Phi.\mathsf{end\text{-}users}$ on behalf of I. Proceed as follows,

1) For $P \in \Phi.\mathsf{end\text{-}users}$, if P is corrupt and sends (`bind`, σ_P) $\xhookrightarrow{t+1} I$ where σ_P is P's signature on (Φ, t), send (`bind`, Φ) $\xhookrightarrow{t+1} \mathcal{F}$ on behalf of P. If P is honest and sends (`bind-ok`) $\overset{t}{\hookrightarrow} \mathcal{F}$, generate P's signature σ_P on (Φ, t), and send it to I on behalf of P.
2) In round $t + 2$, if I sends (`bind-confirmed`) to $\mathcal{F}$, send (`bind`, Φ, t, σ_P, σ_I, σ_Q) $\xhookrightarrow{t+2} \mathcal{C}(\Phi.\mathsf{c_a})$ on behalf of I.

Case I is corrupt

Upon I sending (`bind`, Φ, t, σ_I) $\overset{t}{\hookrightarrow} P \in \Phi.\mathsf{end\text{-}users}$ and P is honest, send (`bind`, Φ) $\overset{t}{\hookrightarrow} \mathcal{F}$ on behalf of I, and send (`send-req`, P) to $\mathcal{F}$. Proceed as follows,

1) For $P \in \Phi.\mathsf{end\text{-}users}$, if P is corrupt, send (`bind-ok`) to $\mathcal{F}$ in round $t + 1$ on behalf of P. Else, if P is honest and sends (`bind-ok`) to $\mathcal{F}$, generate P's signature σ_P on (Φ, t) and send (`bind`, σ_P) to I on behalf of P.
2) In round $t+2$, if I sends (`bind`, Φ, t, Σ) to $\mathcal{C}(\Phi.\mathsf{c_a})$ where Σ is the set of users' signatures on (Φ, t), (`bind-confirmed`) $\xhookrightarrow{t+2} \mathcal{F}$ on behalf of I.

(D) **Bind Update**

Case I is honest

Upon I sending (`bind-update`, $\widetilde{id}$, $\tilde{\theta}$) $\overset{t}{\hookrightarrow} \mathcal{F}$, let $\Phi := \Lambda^I(\widetilde{id})$, generate I's signature σ_I on $msg = (\Phi^I.\mathsf{version} + 1, \Phi^I.\mathsf{shift} + \tilde{\theta}, \tilde{\theta})$, send (`bind-update`, $\widetilde{id}$, msg, σ_I) $\overset{t}{\hookrightarrow} \Phi.\mathsf{end\text{-}users}$ on behalf of I, and proceed as follows,

1) For $P \in \Phi.\mathsf{end\text{-}users}$, distinguish the following cases,
   a) If P is honest and sends (`bind-update-ok`) $\overset{t}{\hookrightarrow} \mathcal{F}$, generate P's signature σ_P on msg send (`bind-update`, $\widetilde{id}$, σ_P) $\xhookrightarrow{t+1} \Phi.\mathsf{other\text{-}users}(P)$ on behalf of P, and forward it to the other end user on behalf of I. In round $t + 2$, if P receives the other end user's signature, forward it to I on behalf of P.
   b) If P is corrupt and sends her signature on msg to I in round $t + 1$, forward it to the other end user on behalf of I. If P's signature is sent to I before round $t + 2$, send (`bind-update-ok`) to $\mathcal{F}$ on behalf of P.
2) In round $t + 3$, if I does not collect others' signatures, initiate the unbinding in procedure (E) on behalf of I. Otherwise, for $\gamma \in \Phi.\mathsf{channels}$:
   a) Update I's balance by $\tilde{\theta}$ and denote the updated channel balance allocation as $\gamma^I.\mathsf{balance}'$, add 1 to $\gamma^I.\mathsf{biList}(\widetilde{id})$ (set to 1 if $\widetilde{id}$ is not in the list) and denote it as $\gamma^I.\mathsf{biList}'$. Generate I's signature σ_I on $msg = (\gamma^I.\mathsf{version}+1, \gamma^I.\mathsf{balance}'$, $\gamma^I.\mathsf{biList}')$, and send σ_I to the other channel user, denoted as P, on behalf of I.
   b) In round $t + 4$, if P is honest, generate P's signature on msg and send it to I on behalf of P. If P is corrupt and does not send P's signature on msg to I, send (`bind-chan-not-updated`, $\gamma.\mathsf{id}$) to $\mathcal{F}$ on behalf of P, and initiate channel closing in procedure (E) on behalf of I.

Fig. 15: Simulation: (A) Open, (B) Channel Update, (C) Bind, and (D) Bind Update.

We abbreviate $I := \Phi.\mathsf{Irene}$ and $Q := \beta.\mathsf{other\text{-}user}(P)$ for $P \in \beta.\mathsf{users}$.

(D) **Bind Update**

Case I is corrupt

Upon I sending (`bind-update`, $\widetilde{id}$, *msg* $= (\mathit{ver}, \mathit{shift}, \tilde{\theta})$, σ_I) $\overset{t}{\hookrightarrow} P \in \Phi.\mathsf{end\text{-}users}$, send (`bind-update`, $\widetilde{id}$, $\tilde{\theta}$) $\overset{t}{\hookrightarrow} \mathcal{F}$ on behalf of I, and (`send-req`, P) $\overset{t}{\hookrightarrow} \mathcal{F}$. Proceed as follows,

1) For $P \in \Phi.\mathsf{end\text{-}users}$, distinguish the following cases,
   a) If P is honest and sends (`bind-update-ok`) $\overset{t+1}{\hookrightarrow} \mathcal{F}$, generate P's signature σ_P on *msg* and send (`bind-update`, $\widetilde{id}$, σ_P) $\overset{t+1}{\hookrightarrow} \Phi.\mathsf{other\text{-}users}(P)$ on behalf of P. If P receives the other end user's signature on *msg* in round $t+2$, forward the message to I on behalf of P.
   b) If P is corrupt and P's signature is sent to the honest end user before round $t+2$, send (`bind-update-ok`) to $\mathcal{F}$ on behalf of P.
2) If the honest users do not collect others' signatures, initiate unbinding on behalf of the honest users in procedure (E). Otherwise, for each honest user P, in his channel $\gamma \in \Phi.\mathsf{channels}$, proceed as follows,
   a) Update I's balance by $\tilde{\theta}$ and denote the updated channel balance allocation as $\gamma^I.\mathsf{balance}'$, add 1 to $\gamma^I.\mathsf{biList}(\widetilde{id})$ (set to 1 if $\widetilde{id}$ is not in the list) and denote it as $\gamma^I.\mathsf{biList}'$. Let *msg* $:= (\gamma^I.\mathsf{version} + 1, \gamma^I.\mathsf{balance}', \gamma^I.\mathsf{biList}')$.
   b) If I sends σ_I to P in round $t+3$ where σ_I is I's signature on *msg*, generate P's signature σ_P on *msg*, and send σ_P to I in round $t+4$ on behalf of P. Otherwise, (`bind-chan-not-updated`, $\gamma.\mathsf{id}$) $\overset{t+3}{\hookrightarrow} \mathcal{F}$ on behalf of I, and initiates channel closing in procedure (F) on behalf of P.

(E) **Unbind**

1) Distinguish the following cases,
   a) If P is honest, upon P sending (`unbind`, $\widetilde{id}$) $\overset{t}{\hookrightarrow} \mathcal{F}$ (let $\Phi := \Lambda^P(\widetilde{id})$), or be invoked by other procedures (let t be the current round), (`unbind`, $\widetilde{id}$, $\Phi^P.\mathsf{state}$) $\overset{t}{\hookrightarrow} \mathcal{C}(\Phi.\mathsf{c_a})$ on behalf of P. Denote it reaches $\mathcal{L}$ in round $\tau \leq t + \Delta$.
   b) If P is corrupt, upon P sending (`unbind`, $\widetilde{id}$, W) $\overset{t}{\hookrightarrow} \mathcal{C}(\Phi.\mathsf{c_a})$, (`unbind`, $\widetilde{id}$) $\overset{t}{\hookrightarrow} \mathcal{F}$ on behalf of P. Denote it reaches $\mathcal{L}$ in round $\tau \leq t + \Delta$. If the pending update is confirmed in W, send confirmation to $\mathcal{F}$ on behalf of P.
2) For $Q \in \Phi.\mathsf{other\text{-}users}(P)$, distinguish the following cases,
   a) If Q is honest, send (`unbind`, $\widetilde{id}$, $\Phi^Q.\mathsf{state}$) $\overset{\tau}{\hookrightarrow} \mathcal{C}(\Phi.\mathsf{c_a})$ on behalf of Q.
   b) If Q is corrupt and sends a state that confirms the pending state, send confirmation to $\mathcal{F}$ on behalf of Q.
3) If any corrupt user does not send message to $\mathcal{C}(\Phi.\mathsf{c_a})$, send (`timeout`, $\widetilde{id}$) $\overset{\tau+\Delta}{\hookrightarrow} \mathcal{C}(\Phi.\mathsf{c_a})$ on behalf of honest users.
4) If the pending update is confirmed, for channel $\gamma \in \Phi.\mathsf{channels}$ that has not been initiated closing: go to step (2) of the simulation on procedure (D).

(F) **Close**

Case P is honest and Q is corrupt

Upon P sending (`close`, *id*) $\overset{t}{\hookrightarrow} \mathcal{F}$ (let $\beta := \Gamma^P(id)$), or be invoked by other procedures (let t be the current round), (`close`, $\beta^P.\mathsf{state}$) $\overset{t}{\hookrightarrow} \mathcal{C}(\beta)$ on behalf of P. Assume it reaches $\mathcal{L}$ in round $\tau \leq t + \Delta$. If Q sends a state with the pending update confirmed to $\mathcal{C}(\beta)$ in round τ, send the confirmation to $\mathcal{F}$ on behalf of Q. Distinguish the following cases,

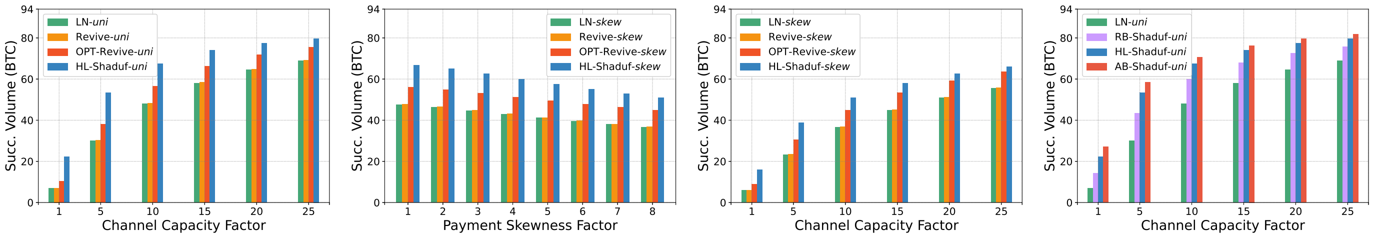
1) If $\beta.\mathsf{bind} = \emptyset$ and Q sends no message to $\mathcal{C}(\beta)$, send (`close-resolve`) to $\mathcal{C}(\beta)$ on behalf of P in round $\tau + \Delta$.
2) Otherwise, proceed as follows,
   a) For each bind in $\beta.\mathsf{bind}$, proceed procedure (E) in round t taking P as the initiator.
   b) Let $\tau_1 \leq t + 3\Delta$ be the round when all bindings are unbound, send (`close-resolve`) $\overset{\tau_1}{\hookrightarrow} \mathcal{C}(\beta)$ on behalf of P.

Case P is corrupt and Q is honest

Upon P sending (`close`, W) $\overset{t}{\hookrightarrow} \mathcal{C}(\beta)$, send (`close`, $\beta.\mathsf{id}$) $\overset{t}{\hookrightarrow} \mathcal{F}$ on behalf of P. Assume it reaches $\mathcal{L}$ in round $\tau \leq t + \Delta$. If W confirms the pending channel update, send confirmation to $\mathcal{F}$ on behalf of P. Distinguish the following cases,

1) If $\beta.\mathsf{bind} = \emptyset$, send (`close`, $\beta^Q.\mathsf{state}$) $\overset{\tau}{\hookrightarrow} \mathcal{C}(\beta)$ on behalf of Q.
2) Otherwise, proceed as follows,
   a) For each bind in $\beta.\mathsf{bind}$, if the unbinding has not been initiated in round τ, go to procedure (E) taking Q as the initiator.
   b) Let $\tau_1 \leq \tau + 3\Delta$ be the round when all bindings are unbound, send (`close-resolve`) $\overset{\tau_1}{\hookrightarrow} \beta$ on behalf of Q.

Fig. 16: Simulation: (D) Bind Update, (E) Unbind, and (F) Close.



(a) Comparing succ. volume of 4 schemes under uniform payment demands, with channel capacity factor from 1 to 25.

(b) Comparing succ. volume of 4 schemes under skewed payment demands, with capacity factor 10 and payment skewness factor from 1 to 8.

(c) Comparing succ. volume of 4 schemes under skewed payment demands, with skewness factor 8 and channel capacity factor from 1 to 25.

(d) Comparing succ. volume of LN and 3 binding strategies of Shaduf under uniform payment demands, with channel capacity factor from 1 to 25.

Fig. 17: Success volume of the evaluation.