FlexiPCN: Flexible Payment Channel Network

Susil Kumar Mohanty and Somanath Tripathy

Department of Computer Science and Engineering
Indian Institute of Technology Patna, Bihta, Patna, India, 801106
`susil_1921cs05@iitp.ac.in` and `som@iitp.ac.in`

Abstract. Payment Channel Network (PCN) is a widely recognized and effective off-chain solution used to reduce on-chain operational costs. PCN is designed to address the scalability challenge and throughput issues in permissionless blockchains. Though transaction throughput is improved, many issues remain, like no flexibility, channel exhaustion, poor sustainability, etc. A separate deposit is required for each payment channel between two users, which locks a substantial amount of coins for a long period of time. Therefore, the flexibility to move these locked coins across channels is impossible through off-chain. Moreover, the channels get exhausted due to unbalanced (unidirectional) transfer. This causes the channel to become unsustainable (dead) until the PCN is rebalanced. This work presents a novel payment protocol called Flexible Payment Channel Networks (FlexiPCN), which allows users to deposit coins per user rather than per channel. So, users can move coins flexibly from one channel to another without the help of the blockchain or setting the cycle off-chain. FlexiPCN has been proven to be secure under the Universal Composability framework.

Keywords: Blockchain · Payment channel network · Security · Privacy

1 Introduction

In the past decade, blockchain technology has rapidly developed, making it possible to conduct secure transactions in a distributed and trustless environment [17, 24]. It supports complex transaction logic using smart contracts [22], which are a few lines of code that run on the blockchain [24]. It is a robust technology because of its consensus mechanism (like PoW[1]), which allows all peers to have a consistent view of transactions [17]. However, its wide adoption is limited due to scalability issues (low transaction throughput, high transaction fees, and high latency) [3,19]. For example, Bitcoin [17] executes 5-7 transactions per second (TPS) and takes approximately 60 minutes to finalize a transaction. Similarly, Ethereum [24] executes 15-20 TPS and takes approximately 6 minutes to finalize a transaction. However, traditional payment systems like Visa process about 47, 000 TPS[2]. To solve the scalability issue, a novel off-chain (layer 2)

[1]Adam Back. "HashCash: A popular PoW system". First announced in March 1997.

[2]Stress Test Prepares VisaNet for the Most Wonderful Time of the Year. https://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html

mechanism known as payment channels has been introduced [19]. Payment channel enables two users to deposit and lock funds in the blockchain, as well as make payments without broadcasting or recording transactions on the blockchain. Off-chain payments are processed and confirmed instantly since they only need the consent of channel users instead of all peers on the blockchain. If any dispute arises between the channel users, it is resolved through the blockchain. Payment channels are extended to the payment channel network (PCN), allowing users without a direct payment channel to perform multi-hop payments without creating a new payment channel. For example, the Lightning Network (LN) [19] and Raiden Network (RN) [1] are payment channel networks deployed on top of Bitcoin and Ethereum, respectively. For more information about PCN, see [4,6], and related security and privacy issues [7,8,15,18,21].

Even though PCN has better scalability, it still has many problems. In PCNs, users must deposit separately for each channel [14], and significant amounts of coins are locked up in advance. Payment flows over a channel are not equal in both directions, so funds accumulate gradually in one direction. Consequently, over time, the balance in one direction of channels gradually becomes exhausted because of imbalanced channel transfers [20]. In such a case, the following issues may arise:

1. Any further payments are not processed because there are insufficient funds in the required channel.
2. It must either be revoked from the blockchain or refunded by closing the payment channel and then reopening it. These are on-chain operations, which are time-consuming and costly.
3. Whenever a channel is exhausted, owners lose the opportunity to receive off-chain payments as relay fees.
4. PCN payment routing becomes more difficult because the user must find a payment path with enough capacity. Also, the success rate of probing may decrease.

Therefore, channel exhaustion is an important issue in PCN.

There are several existing works that address this issue. The trivial approach is to refund the channel by closing and reopening the channel, which requires two on-chain transactions that are costly and time-consuming. LOOP[3] reduces the refund costs to one on-chain transaction. Refunding still needs to interact with the blockchain. Another approach is Revive [10], where channels are refunded by reallocating deposits from adjacent channels, known as "rebalancing". The entire process is off-chain, so rebalancing is cost-free for multiple times, and the underlying blockchain is relieved of the transaction load. The rebalancing operation works well in PCN; provided i) when both channel users who wish to rebalance and the direction of desired coin flows can form directed cycles; ii) a fair leader is required to collect users' rebalancing demands, identify the directed cycles, and generate transactions for cycles; and iii) the cycle users should cooperate. Finally, a minimum rebalancing amount can be achieved among the cycle users.

[3] `https://lightning.engineering/loop/`

Because of this, it suffers from low feasibility for large-scale applications, such as LN [19]. PnP [11] is another solution that relies on carefully planning the initial balance on each channel to reduce the chance of channel exhaustion. It delays the occurrence of channel exhaustion by guaranteeing a good probability of success for off-chain transactions. It is very difficult to estimate node-to-node payment requirement correctly, a prior. Since PCN is trustless by nature, malicious nodes may compromise the balance planning service. PnP [11] cannot recover nearly exhausted channels. CYCLE [9] is an asynchronous rebalancing approach for sustainable PCN that allows the channels to rebalance during off-chain payment execution. It consistently balances PCN channels and prevents channel freezing while ensuring privacy and security of users. Shaduf [5], is a payment channel rebalancing scheme that doesn't require any cycles. It allows users to shift coins off-chain, several times after an on-chain binding operation is performed. The binding process is an on-chain operation, so it becomes time-consuming and expensive. Thus, channel exhaustion issue in PCN has not yet been fully resolved by the off-chain method. It is also not flexible because funds are locked into the blockchain for a specific channel.

This work introduces a flexible payment channel network called FlexiPCN, a purely off-chain based rebalancing technique, which allows users to freely allocate and share funds across all of their payment channels. It keeps funds per user rather than per channel, facilitate users to use funds more flexibly and improves payment success rates. Therefore, channel exhaustion only occurs when the user has exhausted all its funds or if the payment amount exceeds their current balance.

The rest of this paper is structured as follows. Section 2 explains the background concept. System model and formal model is described in Section 3, and the proposed FlexiPCN method is described in Section 4. Security analysis is presented in Section 5. Section 6 discusses the conclusion and future scope of the paper.

2 Background

Payment Channel Network (PCN): A payment channel enables several payments to be made between two parties without recording each transaction to the blockchain. A PCN is made up of peer-to-peer multi-hop payment channels. It is still possible to conduct payments via PCN even if there is no direct channel between two peers. It is created when two peers deposit coins into a shared account and add double-signed transactions to a blockchain. When the channel does not require or one party's coins run out, coins are distributed based on its final state, and closes the channel is recorded on the blockchain. The deposit will be refunded to each user according to their mutually agreed channel state. A PCN is made up of peer-to-peer multi-hop payment channels. It is still possible to conduct payments via PCN even if there is no direct channel between two peers. For more information about PCN, see [12, 13, 16].

Off-chain Contracts: Off-chain contracts [19] are smart contracts in which the contract logic is not executed by miners. It is carried out by all participants involved in making the contract. It is possible to execute computationally intensive operations without involving the blockchain, so long as all participants are honest. An honest participant can prove the correct state of the contract. It is impossible to cheat because all the participants must sign a contract. Whenever a malicious participant broadcasts an incorrect state in the blockchain, the counterparty can dispute it and broadcast the correct state. An example of such contact is HTLC[4], which is used in PCN.

Hashed Time-Lock Contracts (HTLC): An off-chain payment transferred from sender to receiver must be atomic. Either the payment channel balances are entirely updated or terminated. PCN [19] accomplishes atomicity by incorporating Hashed Time-Lock Contracts (HTLC)[4]. For more information about HTLC, see [12, 13, 16, 23]. To build an HTLC contract, the receiver first chooses a random string and then transmits the hash of the string to the sender. The sender computes the total amount (including forwarding fees), time-lock for all the channels on the path. Next, it locks funds and sends payment to the subsequent user. During the locking phase, each intermediary user forwards the payment to the next neighbor along the path by deducting time-lock and forwarding fees from its preceding user's payment request. During the releasing phase, upon receiving the preimage from its right neighbor within the time-lock, it releases the coin to the next neighbor and reveals the preimage to its previous neighbor. Any disagreement would be handled through blockchain. For more information about HTLC, see [13, 16].

Table 1: Notations

Notation	Description	Notation	Description
$\mathbb{G} := (\mathbb{V}, \mathbb{E}, \Omega)$	Payment channel network	ω_i	User u_i's collaterals or sum of neighboring channel's collateral
$c_{\langle u_i, u_j \rangle}$	Channel identifier	v	Actual amount sender u_0 wants to transfer to receiver u_n
$\mathcal{B}$	Blockchain	Υ_{ij}	Collateral or available balance in channel $c_{\langle u_i, u_j \rangle}$ $(u_i \rightarrow u_j)$
$\mathcal{G}$	Elliptic curve base point	$c_{\langle u_i, u_j \rangle}$	Payment channel identifier (between user u_i and u_j)
$\mathcal{P}$	Payment path	t_{ij}	Expiration time of the transaction corresponding to user u_i
u_0 & u_n	Sender & Receiver	f_{ij}	Payment relay fee of channel $c_{\langle u_i, u_j \rangle}$
$\{u_i\}_{i \in [1, n-1]}$	Intermediate users	sk_i & pk_i	Secret key of user u_i & Public key of user u_i
$\mathcal{A}$	Attacker	$\mathcal{AL}$ & $\mathcal{CL}$	Active channel list & Closing channel list
$\mathcal{F}$	Ideal functionality	PK, R, & S	Aggregated public key, partial nonce, & signature
$\mathtt{C}_{ij}$	Channel capacity	$\mathcal{B}[u_i]$	On-chain balance of the user u_i
$\mathcal{T}$	Coin Allocation Table	$\nu_{\mathtt{ij}}$	The amount required to shift for fulfilling the payment request
$\mathcal{T}_i$	Transaction log of user u_i	σ_{ij}	Signature signed by the user u_i and send its neighbor u_j

[4]https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts

3 System and Formal Model

3.1 System Model

We model FlexiPCN as a bi-directed and weighted graph $\mathbb{G} := (\mathbb{V}, \mathbb{E}, \Omega)$, where $\mathbb{V}$ denotes the set of blockchain users (nodes) with a weight function w, $w : \mathbb{V} \times \mathbb{V} \to \mathbb{R}^+$ denotes the collateral between users, $\mathbb{E} \subseteq \mathbb{V} \times \mathbb{V}$ denotes the set of active payment channels between two user's wallets, and $\omega_u \in \Omega$ is the collateral of user u. Each user $u_i in \mathbb{V}$ has a collateral of $\omega_i = \sum_{j=1}^{k} \Upsilon_{ij}$, where Υ_{ij} represents the collateral balance of u_i with u_j. An off-chain payment is denoted as $\texttt{Payment}(u_0, u_n, v_{01}, t_{01}, \mathcal{P})$, where u_0 is the sender, u_n is the receiver, $v_{01} = v + \sum_{i=1}^{n-1} f_{\langle i,i+1 \rangle}$ is the total payment amount including relaying fees, and t_{01} is the time period within which the payment must be completed. $f_{\langle i,i+1 \rangle}$ represents the relay fee that an intermediate user $u_i \in \mathbb{V}$ charges for relaying a payment from u_i to u_{i+1}. A payment path $\mathcal{P}$ between u_0 and u_n is denoted as $\{u_0 \to u_1 \to \cdots \to u_n\}$. For readability, we present the most frequently used notations in Table 1.

3.2 Adversary Model

We assume that each pair of users uses a secure and authenticated channel to exchange payment information. Neither the sender nor the receiver have a secure channel with the intermediate users, but both have a secure channel with each other. Intermediary nodes are only aware of their previous and next neighbors. The sender, however, has detailed information about the network topology, including the identity of the user, lock-time, relay fees of the intermediate nodes, channel identifier, and node capacity but not the channel capacity, etc. There must be at least one payment path between the sender and receiver, and the associated payment channels must meet the collateral requirements (either from one channel or by moving coins from other channels) to process off-chain transactions. Each neighboring user has a pre-established payment channel with zero collateral.

Security threats occur either from internal or from external sources. So, our adversary model considers both inside and outside PCN adversaries. We consider both honest-but-curious and malicious models. Assume that $\mathcal{A}$ is a computationally efficient adversary who corrupts one or more PCN users. Once $\mathcal{A}$ has corrupted some users, it gains access to their internal states and information flows, as well as complete control over them. Compromised users can work together to challenge PCN security, and steals off-chain payment relay fees from honest intermediaries. $\mathcal{A}$ can impersonate any corrupted user and sends arbitrary messages. Assume that intermediate users are honest-but-curious and that they are interested in analyzing the sender's privacy and communication patterns or behaviors. The target of the adversary is to exhaust a payment channel, so that it can no longer participate in the payment execution.

3.3 Ideal World Model

The security model of FlexiPCN follows [5, 12, 16, 23] which is based on the universal composable (UC) framework [2]. The simulator $\mathcal{S}$ is a probabilistic polynomial-time algorithm that simulates the off-chain FlexiPCN protocol in a hybrid world model. The output in $\mathcal{S}$ must be indistinguishable in communication with the ideal functionality $\mathcal{F}$, even if some users are corrupt. The environment $\mathcal{Z}$ represents all events that occur outside the protocol execution, which would influence the protocol execution. It gives the input to the user and obtains the output from the user. $\mathcal{Z}$ could compromise certain users for obtaining access to their internal states and to manage their execution. But, $\mathcal{Z}$ does not interact with $\mathcal{S}$, while getting executed by corrupted users. Since honest users interact through secure and authorized channels, the adversary $\mathcal{A}$ cannot retrieve any confidential information. The ideal functionality $\mathcal{F}_{\texttt{FlexiPCN}}$ uses $\mathcal{F}_{\texttt{ECDSA}}$ (used for digital signatures), $\mathcal{F}_{\mathcal{B}}$ (used to maintain blockchain and its operations), $\mathcal{F}_{\mathcal{C}}$ (used for maintaining contract instances), and $\mathcal{F}_{anon}$ (used for anonymous communication) as subroutines, i.e., our protocol is specified in the $(\mathcal{F}_{\texttt{ECDSA}}, \mathcal{F}_{\mathcal{B}}, \mathcal{F}_{\mathcal{C}}, \mathcal{F}_{\texttt{anon}})$-hybrid model. Internally, $\mathcal{F}_{\texttt{FlexiPCN}}$ maintains two lists, namely an active channel list $\mathcal{AL}$ and a closed channel list $\mathcal{CL}$. It also maintains a table called transaction log or state $\mathcal{T}$.

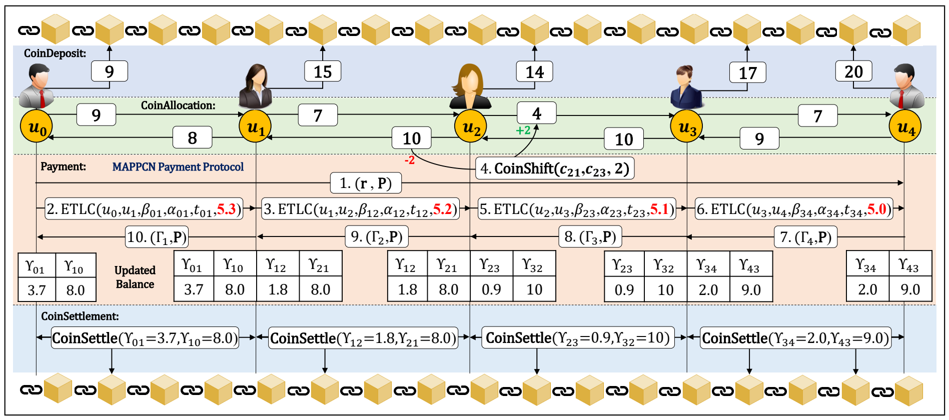

CoinDeposit:
9
15
14
17
20
CoinAllocation:
9
7
4
7
8
10
-2
+2
10
9
u_0
u_1
u_2
u_3
u_4
Payment: MAPPCN Payment Protocol
4. CoinShift(c_{21},c_{23}, 2)
1. (r , P)
2. ETLC(u_0,u_1,β_{01},α_{01},t_{01},5.3)
3. ETLC(u_1,u_2,β_{12},α_{12},t_{12},5.2)
5. ETLC(u_2,u_3,β_{23},α_{23},t_{23},5.1)
6. ETLC(u_3,u_4,β_{34},α_{34},t_{34},5.0)
10. (Γ_1,P)
9. (Γ_2,P)
8. (Γ_3,P)
7. (Γ_4,P)
Updated Balance
Υ_{01} Υ_{10} 3.7 8.0
Υ_{01} Υ_{10} Υ_{12} Υ_{21} 3.7 8.0 1.8 8.0
Υ_{12} Υ_{21} Υ_{23} Υ_{32} 1.8 8.0 0.9 10
Υ_{23} Υ_{32} Υ_{34} Υ_{43} 0.9 10 2.0 9.0
Υ_{34} Υ_{43} 2.0 9.0
CoinSettlement:
CoinSettle(Υ_{01}=3.7,Υ_{10}=8.0)
CoinSettle(Υ_{12}=1.8,Υ_{21}=8.0)
CoinSettle(Υ_{23}=0.9,Υ_{32}=10)
CoinSettle(Υ_{34}=2.0,Υ_{43}=9.0)

Fig. 1: FlexiPCN: Flexible Payment Channel Network Overview

4 The Proposed Protocol: FlexiPCN

4.1 Overview

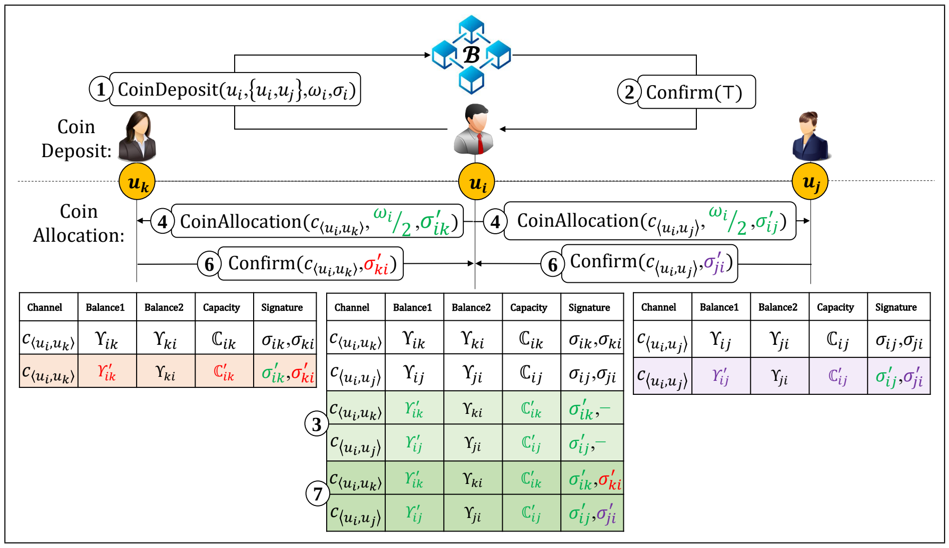
We provide an overview of payment execution in FlexiPCN. Consider the simple PCN structure shown in Fig. 1, in which payment channels are established between users with no collateral. In order to participate in off-chain payment execution, each user deposits a certain amount of coins to the blockchain $\mathcal{B}$, such as 9, 15, 14, 17, and 20 coins deposited by u_0, u_1, u_2, u_3, and u_4, respectively. Then, each user distributes the deposited coins to their corresponding payment channels, such as u_0 allocates 9 coins to channel $c_{\langle u_0,u_1\rangle}$; u_1 allocates 8 and 7 coins to channel $c_{\langle u_1,u_0\rangle}$ and $c_{\langle u_1,u_2\rangle}$, respectively; u_2 allocates 10 and 4 coins to channel $c_{\langle u_2,u_1\rangle}$ and $c_{\langle u_2,u_3\rangle}$, respectively; u_3 allocates 10 and 7 coins to channel $c_{\langle u_3,u_2\rangle}$ and $c_{\langle u_3,u_4\rangle}$, respectively; u_4 allocates 9 coins to channel $c_{\langle u_4,u_3\rangle}$. Note that users are aware of all other users' node balances, but are only aware of the channel balances of their previous and next users. Suppose u_0 is the sender who wishes to send 5 coins to the receiver u_4. For that, u_0 chooses a payment path $\mathcal{P} : \{u_0 \rightarrow u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4\}$. For the payment operation, we use the existing protocol MAPPCN [23] to ensure user privacy and anonymity. u_0 initiates the payment operation by sending $\langle r, P\rangle$ to u_4 over a secure channel, where r is a random number and P is the base point of the Elliptic curve $\mathbb{E}_p$. Then, u_0 establishes an $\texttt{ETLC}(\texttt{u}_0, \texttt{u}_1, \beta_{01}, \alpha_{01}, \texttt{t}_{01}, \texttt{v}_{01})$[5] contract with u_1, where β_{01} and α_{01} are the secret parameters, t_{01} is the expiration time period to lock v_{01} amount of coins ($v_{01} = 5.3$ is the sum of payment amount $v = 5$ and relay fees of intermediate users $f = \sum_{i=1}^{3} f_{\langle i,i+1\rangle} = 0.3$). Later, each intermediate user u_i generates ℓ_i randomly, computes secret parameters $\beta_{\langle i,i+1\rangle} = \ell_i \cdot \beta_{\langle i-1,i\rangle}$ and $\alpha_{\langle i,i+1\rangle} = \ell_i \cdot \alpha_{\langle i-1,i\rangle}$ to establish an $\texttt{ETLC}(\texttt{u}_\texttt{i}, \texttt{u}_{\texttt{i}+1}, \beta_{\langle \texttt{i},\texttt{i}+1\rangle}, \alpha_{\langle \texttt{i},\texttt{i}+1\rangle}, \texttt{t}_{\langle \texttt{i},\texttt{i}+1\rangle}, \texttt{v}_{\langle \texttt{i},\texttt{i}+1\rangle})$ contract. When u_2 receives the ETLC request from u_1, it is unable to forward it to u_3 due to insufficient balance at channel $c_{23} = 4$, which requires at least 5.1 coins after deducting the relay fee. Therefore, u_2 performs a coin shifting operation to shift 2 coins from channel c_{21} to channel c_{23} in order to complete the payment execution. Then, u_2 establishes an $\texttt{ETLC}(\texttt{u}_2, \texttt{u}_3, \beta_{23}, \alpha_{23}, \texttt{t}_{23}, \texttt{v}_{23})$ contract with u_3, which continues until u_4 is reached. Therefore, upon receiving $\texttt{ETLC}(\texttt{u}_3, \texttt{u}_4, \beta_{34}, \alpha_{34}, \texttt{t}_{34}, \texttt{v}_{34})$ request from u_3, u_4 validates $r \cdot \beta_{34} \cdot P \overset{?}{=} \alpha_{34}$, computes $\mathit{\Gamma}_{34} = r \cdot \beta_{34}$, and returns $\langle \mathit{\Gamma}_4, P\rangle$ to u_3 in order to satisfy the ETLC contract condition. After receiving $\langle \mathit{\Gamma}_{i+1}, P\rangle$ from u_{i+1}, each intermediate user u_i validate $\mathit{\Gamma}_{i+1} \cdot P \overset{?}{=} \alpha_{\langle i,i+1\rangle}$. Upon satisfying the contract condition, u_i releases the locked coins to u_{i+1} and returns $\mathit{\Gamma}_i = \ell_i^{-1} \cdot \mathit{\Gamma}_{i+1}$ to u_{i-1} in order to satisfy the ETLC contract condition. As a result, each user on the payment path receives their committed coins. When a user wants to settle coins, it must first finish all pending payments and share the latest state with its neighbors. Then it invokes coin settlement on the blockchain $\mathcal{B}$ by submitting the latest state with its signature. The user account is then updated by $\mathcal{B}$. If a neighbor raises a dispute, the neighbor user sends the most recent state to $\mathcal{B}$, and $\mathcal{B}$ is solved as a dispute resolution.

4.2 FlexiPCN Operations:

FlexiPCN consists of seven primary operations: 1) open channel, 2) coin deposit, 3) coin allocation, 4) payment, 5) coin shift, 6) coin settlement, and 7) close channel.

[5] ETLC: Elliptic Curve based Time-Lock Contract [23]

1. `OpenChannel` : This is an on-chain (blockchain) operation that is triggered by the user u_i in order to establish a payment channel with the user u_j, and it returns the channel identifier $\mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}$.
2. `CoinDeposit` : This is an on-chain operation initiated by user u_i, which deposits the amount of coins ω_i with signature σ_i in order to participate in the payment execution and returns $\top$ as confirmation.
3. `CoinAllocation` : This is an off-chain operation initiated by the user u_i that distributes coins equally to each active adjacent channel user u_j so that it can participate in payment execution. Each adjacent user returns $\sigma^{'}_{ji}$ as their agreement confirmation.
4. `Payment` : This is an off-chain operation initiated by the sender u_0 who wants to transfer some coins (v) to u_n via some intermediate users. This is accomplished by using the ETLC[5] contract, which updates the channel balances atomically.
5. `CoinShift` : This is an off-chain operation triggered by an intermediate user u_i between two channels that do not have enough channel balance to send the ETLC payment request to the next user u_j. In order to fulfill the payment request, u_i moves the required amount of coins ν from its adjacent channel(s) to another.
6. `CoinSettlement` : This is an on-chain operation initiated by any user u_i, by sending the recently updated state Ψ with its agreement σ_i for coin settlement after all pending payment requests are completed.
7. `CloseChannel` : This is an on-chain operation that can be initiated by any channel user to close the payment channel and return the off-chain balance to the blockchain.



$\mathcal{B}$
1 CoinDeposit(u_i,{u_i,u_j},ω_i,σ_i)
2 Confirm(⊤)
Coin Deposit:
u_k
u_i
u_j
Coin Allocation:
4 CoinAllocation($c_{\langle u_i,u_k\rangle}$, $\omega_i/2$,σ'_{ik})
4 CoinAllocation($c_{\langle u_i,u_j\rangle}$, $\omega_i/2$,σ'_{ij})
6 Confirm($c_{\langle u_i,u_k\rangle}$,$\sigma'_{ki}$)
6 Confirm($c_{\langle u_i,u_j\rangle}$,$\sigma'_{ji}$)

Channel	Balance1	Balance2	Capacity	Signature
$c_{\langle u_i,u_k\rangle}$	Υ_{ik}	Υ_{ki}	$\mathbb{C}_{ik}$	σ_{ik},σ_{ki}
$c_{\langle u_i,u_k\rangle}$	Υ'_{ik}	Υ_{ki}	$\mathbb{C}'_{ik}$	σ'_{ik},σ'_{ki}

	Channel	Balance1	Balance2	Capacity	Signature
	$c_{\langle u_i,u_k\rangle}$	Υ_{ik}	Υ_{ki}	$\mathbb{C}_{ik}$	σ_{ik},σ_{ki}
	$c_{\langle u_i,u_j\rangle}$	Υ_{ij}	Υ_{ji}	$\mathbb{C}_{ij}$	σ_{ij},σ_{ji}
3	$c_{\langle u_i,u_k\rangle}$	Υ'_{ik}	Υ_{ki}	$\mathbb{C}'_{ik}$	σ'_{ik},–
	$c_{\langle u_i,u_j\rangle}$	Υ'_{ij}	Υ_{ji}	$\mathbb{C}'_{ij}$	σ'_{ij},–
7	$c_{\langle u_i,u_k\rangle}$	Υ'_{ik}	Υ_{ki}	$\mathbb{C}'_{ik}$	σ'_{ik},σ'_{ki}
	$c_{\langle u_i,u_j\rangle}$	Υ'_{ij}	Υ_{ji}	$\mathbb{C}'_{ij}$	σ'_{ij},σ'_{ji}

Channel	Balance1	Balance2	Capacity	Signature
$c_{\langle u_i,u_j\rangle}$	Υ_{ij}	Υ_{ji}	$\mathbb{C}_{ij}$	σ_{ij},σ_{ji}
$c_{\langle u_i,u_j\rangle}$	Υ'_{ij}	Υ_{ji}	$\mathbb{C}'_{ij}$	σ'_{ij},σ'_{ji}

Fig. 2: Coin Deposit and Allocation Operation

Algorithm 1: FlexiPCN Protocol Operations

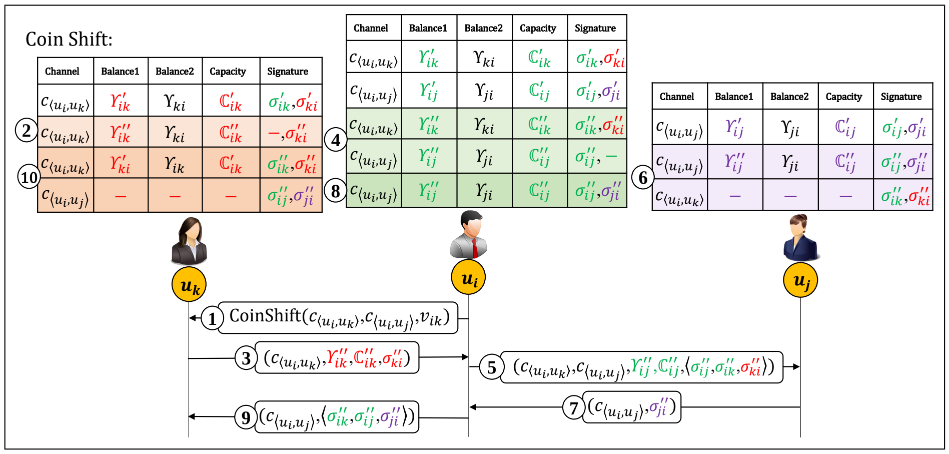
```
OpenChannel($\mathtt{u_i}, \mathtt{u_j}, \Upsilon_{ij}, \Upsilon_{ji}, \mathtt{f_{ij}}, \mathtt{t_{ij}}$) :
1: if $(\{u_i, u_j\} \in \mathcal{B}$ and $c_{\langle u_i,u_j \rangle} \notin \mathcal{AL})$ then
2:     create: channel identifier $\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}$
3:     $\Upsilon_{ij} = 0$, $\Upsilon_{ji} = 0$, $\mathbb{C}_{ij} = 0$
4:     write: open($\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \mathbb{C}_{ij}, \mathtt{f_{ij}}, \mathtt{t_{ij}}$) to $\mathcal{B}$
5:     store: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \Upsilon_{ij}, \Upsilon_{ji}, \mathbb{C}_{ij}, \mathtt{f_{ij}}, \mathtt{t_{ij}})$ in $\mathcal{AL}$
6:     send: $c_{\langle u_i,u_j \rangle}$ to both $u_i$ and $u_j$
7: else
8:     Abort
9: end if

CoinDeposits($\mathtt{u_i}, \{\mathtt{u_j}\}, \omega_i, \sigma_i$) :
1: if $(\mathtt{u_i} \in \mathcal{B}$ and $\omega_i \leq \mathcal{B}[\mathtt{u_i}])$ then
2:     for all neighbor $\mathtt{u_j}$ do
3:         if $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle} \in \mathcal{AL})$ then
4:             add: $\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}$ in $\mathcal{T}$
5:         else
6:             Abort
7:         end if
8:         write: CoinDeposit($\mathtt{u_i}, \{\mathtt{u_j}\}, \omega_i, \sigma_i$) to $\mathcal{B}$
9:         send: $\top$ to $u_i$
10:    end for
11: else
12:    Abort
13: end if

CoinAllocation($\mathtt{u_i}, \{\mathtt{u_j}\}, \omega_i$) :
1: for all neighbor $\mathtt{u_j}$ do
2:     if $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle} \in \mathcal{AL}$ and $\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle} \in \mathcal{T})$ then
3:         update: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \Upsilon'_{ij} = \frac{\omega_i}{|\{\mathtt{u_j}\}|}, \Upsilon_{ji}, \mathbb{C}'_{ij} = \Upsilon'_{ij} + \Upsilon_{ji}, \langle \sigma'_{ij}, - \rangle)$ in $\mathcal{T}$
4:         send: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \frac{\omega_i}{|\{\mathtt{u_j}\}|}, \sigma'_{ij})$ to $u_j$
5:         receive: $\sigma'_{ji}$ from $u_j$
6:         update: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \cdot, \cdot, \cdot, \langle \cdot, \sigma'_{ji} \rangle)$ in $\mathcal{T}$
7:     else
8:         Abort
9:     end if
10:    send: $\top$ to $u_i$
11: end for

Payment: Refer MAPPCN [23].

CoinShift($\{\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_k} \rangle}\}, \mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \nu_{ik}$) :
1: for each channel $\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_k} \rangle}$ do
2:     if $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_k} \rangle} \in \mathcal{AL})$ then
3:         send: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_k} \rangle}, \mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \nu_{ik})$ to $u_k$
4:         receive: $(\Upsilon''_{ik}, \mathtt{C}''_{ik}, \sigma''_{ki})$ from $u_k$
5:         update: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_k} \rangle}, \Upsilon''_{ik}, \cdot, \mathbb{C}''_{ik}, \langle \sigma''_{ik}, \sigma''_{ki} \rangle)$ in $\mathcal{T}$
6:         update: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \Upsilon''_{ij}, \cdot, \mathbb{C}''_{ij}, \langle \sigma''_{ij}, - \rangle)$ in $\mathcal{T}$
7:         send: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_k} \rangle}, \mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \Upsilon''_{ij}, \cdot, \mathbb{C}''_{ij}, \langle \sigma''_{ij}, \sigma''_{ik}, \sigma''_{ki} \rangle)$ to $u_j$
8:         receive: $\sigma''_{ji}$ from $u_j$
9:         update: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \cdot, \cdot, \cdot, \langle \cdot, \sigma''_{ji} \rangle)$ in $\mathcal{T}$
10:        send: $\langle \sigma''_{ik}, \sigma''_{ij}, \sigma''_{ji} \rangle$ to $u_j$
11:    end if
12: end for

CoinSettlement($\mathtt{u_i}, \{\mathtt{u_j}\}, \Psi, \sigma_i$) :
1: send: $(c_{\langle u_i,u_j \rangle}, \Psi)$ to $u_j$
2: write: CoinSettle($\mathtt{u_i}, \Psi, \sigma_i$) to $\mathcal{B}$
3: send: $\top$ to $u_i$

CloseChannel($\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle},$) :
1: if $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \mathbb{C}_{ij}, \mathtt{f_{ij}}, \mathtt{t_{ij}}) \in \mathcal{B}$ and $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \Upsilon_{ij}, \Upsilon_{ji}, \mathbb{C}_{ij}, \mathtt{f_{ij}}, \mathtt{t}'_{ij}) \in \mathcal{AL}$ then
2:     if $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle} \in \mathcal{CL}$ or $\mathtt{t}'_{ij} > |\mathcal{B}|)$ then
3:         Abort
4:     else
5:         remove: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \Upsilon_{ij}, \Upsilon_{ji}, \mathbb{C}_{ij}, f_{ij}, t'_{ij})$ from $\mathcal{AL}$
6:         write: close($\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \mathbb{C}_{ij}, \mathtt{f_{ij}}, t'_{ij}$) to $\mathcal{B}$
7:         store: $\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}$ in $\mathcal{CL}$
8:         send: $(\mathtt{c}_{\langle \mathtt{u_i},\mathtt{u_j} \rangle}, \top)$ to both $\mathtt{u_i}$ and $\mathtt{u_j}$
9:     end if
10: else
11:    Abort
12: end if
```

4.3 FlexiPCN Operational Details:

The open channel and close channel operations are the same as the existing payment protocols like MAPPCN [23], but there is no initial channel balance. However, for the payment operation, we use the existing mechanism of MAPPCN [23] to ensure the anonymity and privacy of the user. Algorithm 1 depicts all the operations of the FlexiPCN protocol. The remaining operations are described in detail as follows:

$\underline{\texttt{CoinDeposit}(\texttt{u}_\texttt{i}, \{\texttt{u}_\texttt{j}\}, \omega_\texttt{i}, \sigma_\texttt{i})}$: As illustrated in Fig. 2, user u_i wishes to deposit some coins onto blockchain $\mathcal{B}$, it first invokes $\texttt{CoinDeposit}(\texttt{u}_\texttt{i}, \{\texttt{u}_\texttt{j}\}, \omega_\texttt{i}, \sigma_\texttt{i})$ and passes the parameters: active neighboring set $\{u_j\}$, amount to be deposited ω_i, and signature σ_i. After receiving $\texttt{CoinDeposit}$ request, the blockchain $\mathcal{B}$ checks $\texttt{u}_\texttt{i} \in \mathcal{B}$, $\texttt{c}_{\langle \texttt{u}_\texttt{i}, \texttt{u}_\texttt{j} \rangle} \in \mathcal{AL}$, and $\omega_i \leq \mathcal{B}[u_i]$. If all of these conditions are met, $\mathcal{B}$ updates the account balance of u_i $\mathcal{B}[u_i] = \mathcal{B}[u_i] - \omega_i$ and sends $\top$ to u_i as confirmation. Then, u_i adds the channel $c_{\langle u_i, u_j \rangle}$ into the transaction state table $\mathcal{T}$.

$\underline{\texttt{CoinAllocation}(\texttt{u}_\texttt{i}, \{\texttt{u}_\texttt{j}\}, \omega_\texttt{i}, \sigma'_\texttt{ij})}$: As illustrated in Fig. 2, the user u_i allocates collateral to each adjacent channel $c_{\langle u_i, u_j \rangle}$ by using the coin allocation policy is $\frac{\omega_\texttt{i}}{|\{\texttt{u}_\texttt{j}\}|}$ along with signature σ'_{ij} for each u_j. After that, each neighbor u_j confirms their agreement by sending their signature σ'_{ji} to u_i. Each user maintains a transaction log (state) $\mathcal{T}$ table.


Coin Shift:
u_k
u_i
u_j
1 CoinShift($c_{\langle u_i,u_k\rangle}$,$c_{\langle u_i,u_j\rangle}$,$v_{ik}$)
3 ($c_{\langle u_i,u_k\rangle}$,$\Upsilon''_{ik}$,$\mathbb{C}''_{ik}$,$\sigma''_{ki}$)
5 ($c_{\langle u_i,u_k\rangle}$,$c_{\langle u_i,u_j\rangle}$,$\Upsilon''_{ij}$,$\mathbb{C}''_{ij}$,$\langle\sigma''_{ij}$,$\sigma''_{ik}$,$\sigma''_{ki}\rangle$)
7 ($c_{\langle u_i,u_j\rangle}$,$\sigma''_{ji}$)
9 ($c_{\langle u_i,u_j\rangle}$,$\langle\sigma''_{ik}$,$\sigma''_{ij}$,$\sigma''_{ji}\rangle$)

	Channel	Balance1	Balance2	Capacity	Signature
	$c_{\langle u_i,u_k\rangle}$	Υ'_{ik}	Υ_{ki}	$\mathbb{C}'_{ik}$	σ'_{ik},σ'_{ki}
2	$c_{\langle u_i,u_k\rangle}$	Υ''_{ik}	Υ_{ki}	$\mathbb{C}''_{ik}$	$-$,σ''_{ki}
10	$c_{\langle u_i,u_k\rangle}$	Υ'_{ki}	Υ_{ik}	$\mathbb{C}'_{ik}$	σ''_{ik},σ''_{ki}
	$c_{\langle u_i,u_j\rangle}$	$-$	$-$	$-$	σ''_{ij},σ''_{ji}

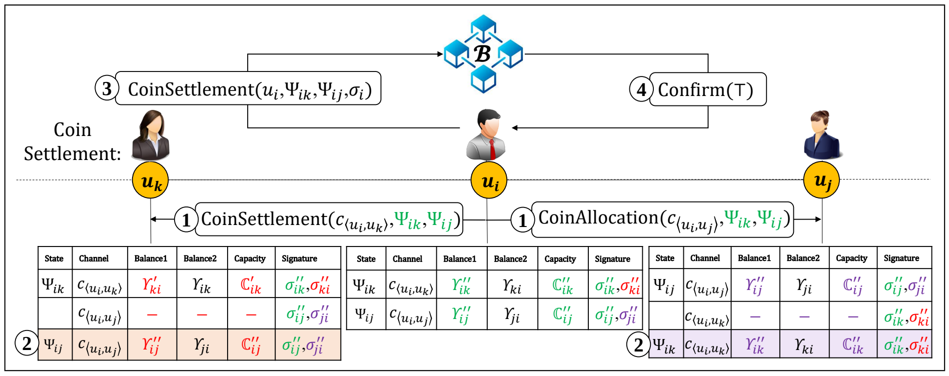
	Channel	Balance1	Balance2	Capacity	Signature
	$c_{\langle u_i,u_k\rangle}$	Υ'_{ik}	Υ_{ki}	$\mathbb{C}'_{ik}$	σ'_{ik},σ'_{ki}
	$c_{\langle u_i,u_j\rangle}$	Υ'_{ij}	Υ_{ji}	$\mathbb{C}'_{ij}$	σ'_{ij},σ'_{ji}
4	$c_{\langle u_i,u_k\rangle}$	Υ''_{ik}	Υ_{ki}	$\mathbb{C}''_{ik}$	σ''_{ik},σ''_{ki}
	$c_{\langle u_i,u_j\rangle}$	Υ''_{ij}	Υ_{ji}	$\mathbb{C}''_{ij}$	σ''_{ij}, $-$
8	$c_{\langle u_i,u_j\rangle}$	Υ''_{ij}	Υ_{ji}	$\mathbb{C}''_{ij}$	σ''_{ij},σ''_{ji}

	Channel	Balance1	Balance2	Capacity	Signature
	$c_{\langle u_i,u_j\rangle}$	Υ'_{ij}	Υ_{ji}	$\mathbb{C}'_{ij}$	σ'_{ij},σ'_{ji}
6	$c_{\langle u_i,u_j\rangle}$	Υ''_{ij}	Υ_{ji}	$\mathbb{C}''_{ij}$	σ''_{ij},σ''_{ji}
	$c_{\langle u_i,u_k\rangle}$	$-$	$-$	$-$	σ''_{ik},σ''_{ki}

Fig. 3: Coin Shift Operation

$\underline{\texttt{CoinShift}(\texttt{c}_{\langle \texttt{u}_\texttt{i}, \texttt{u}_\texttt{k} \rangle}, \texttt{c}_{\langle \texttt{u}_\texttt{i}, \texttt{u}_\texttt{j} \rangle}, \nu_\texttt{ik})}$: As illustrated in Fig. 3, the user u_i moves ν_{ik} amount of coins from one (or more) neighboring channel(s) to $\texttt{c}_{\langle \texttt{u}_\texttt{i}, \texttt{u}_\texttt{j} \rangle}$ in order to fulfill the off-chain payment request routed through it. To do this, u_i sends a $\texttt{CoinShift}(\texttt{c}_{\langle \texttt{u}_\texttt{i}, \texttt{u}_\texttt{k} \rangle}, \texttt{c}_{\langle \texttt{u}_\texttt{i}, \texttt{u}_\texttt{j} \rangle}, \nu_\texttt{ik})$ request to its neighbor u_k (assuming channel $c_{\langle u_i, u_k \rangle}$ has more collateral than ν_{ik} according to $\mathcal{T}$ maintained by u_i). After

u_k updates the channel balance $\Upsilon_{\mathtt{ik}} = \Upsilon_{\mathtt{ik}} - \nu_{\mathtt{ik}}$ and channel capacity $\mathbb{C}_{\mathtt{ik}} = \Upsilon_{\mathtt{ki}} + (\Upsilon_{\mathtt{ik}} - \nu_{\mathtt{ik}})$, u_k sends signature $\sigma_{ki}^{''}$ to u_i. Then, u_i updates the channel balances $\Upsilon_{\mathtt{ik}} = \Upsilon_{\mathtt{ik}} - \nu_{\mathtt{ik}}$ and $\Upsilon_{\mathtt{ij}} = \Upsilon_{\mathtt{ij}} + \nu_{\mathtt{ik}}$, as well as channel capacity $\mathbb{C}_{\mathtt{ik}} = \Upsilon_{\mathtt{ki}} + (\Upsilon_{\mathtt{ik}} - \nu_{\mathtt{ik}})$ and $\mathbb{C}_{\mathtt{ij}} = \Upsilon_{\mathtt{ji}} + (\Upsilon_{\mathtt{ij}} + \nu_{\mathtt{ik}})$, and stores $\sigma_{ki}^{''}$ in $\mathcal{T}$. After that, u_i sends signatures $\langle \sigma_{ij}^{''}, \sigma_{ik}^{''}, \sigma_{ki}^{''} \rangle$ to u_j. Then, u_j updates the channel balance $\Upsilon_{\mathtt{ij}} = \Upsilon_{\mathtt{ij}} + \nu_{\mathtt{ik}}$ and channel capacity $\mathbb{C}_{\mathtt{ij}} = \Upsilon_{\mathtt{ji}} + (\Upsilon_{\mathtt{ij}} + \nu_{\mathtt{ik}})$, stores signatures, and sends $\sigma_{ji}^{''}$ to u_i. Next, u_i stores $\sigma_{ji}^{''}$ and sends $\langle \sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''} \rangle$ to u_k. Finally, all users u_i, u_j, and u_k have the same state.


$\mathcal{B}$
③ CoinSettlement(u_i,Ψ_{ik},Ψ_{ij},σ_i)
④ Confirm(⊤)
Coin Settlement:
u_k
u_i
u_j
① CoinSettlement($c_{\langle u_i,u_k \rangle}$,$\Psi_{ik}$,$\Psi_{ij}$)
① CoinAllocation($c_{\langle u_i,u_j \rangle}$,$\Psi_{ik}$,$\Psi_{ij}$)
②
②

State	Channel	Balance1	Balance2	Capacity	Signature
Ψ_{ik}	$c_{\langle u_i,u_k \rangle}$	Υ_{ki}'	Υ_{ik}	$\mathbb{C}_{ik}'$	$\sigma_{ik}'',\sigma_{ki}''$
	$c_{\langle u_i,u_j \rangle}$	–	–	–	$\sigma_{ij}'',\sigma_{ji}''$
Ψ_{ij}	$c_{\langle u_i,u_j \rangle}$	Υ_{ij}''	Υ_{ji}	$\mathbb{C}_{ij}''$	$\sigma_{ij}'',\sigma_{ji}''$

State	Channel	Balance1	Balance2	Capacity	Signature
Ψ_{ik}	$c_{\langle u_i,u_k \rangle}$	Υ_{ik}''	Υ_{ki}	$\mathbb{C}_{ik}''$	$\sigma_{ik}'',\sigma_{ki}''$
Ψ_{ij}	$c_{\langle u_i,u_j \rangle}$	Υ_{ij}''	Υ_{ji}	$\mathbb{C}_{ij}''$	$\sigma_{ij}'',\sigma_{ji}''$

State	Channel	Balance1	Balance2	Capacity	Signature
Ψ_{ij}	$c_{\langle u_i,u_j \rangle}$	Υ_{ij}''	Υ_{ji}	$\mathbb{C}_{ij}''$	$\sigma_{ij}'',\sigma_{ji}''$
	$c_{\langle u_i,u_k \rangle}$	–	–	–	$\sigma_{ik}'',\sigma_{ki}''$
Ψ_{ik}	$c_{\langle u_i,u_k \rangle}$	Υ_{ik}''	Υ_{ki}	$\mathbb{C}_{ik}''$	$\sigma_{ik}'',\sigma_{ki}''$

Fig. 4: Coin Settlement Operation

$\underline{\mathtt{CoinSettlement}(\mathtt{u_i}, \{\mathtt{u_j}\}, \Psi, \sigma_\mathtt{i}) :}$ As illustrated in Fig. 4, the user u_i informs each neighbor u_j of their most recent updated state $\langle c_{\langle u_i,u_j \rangle}, \Psi_{ij} \rangle$ before initiating the coin settlement operation. Then, u_i writes $\mathtt{CoinSettle}(\mathtt{u_i}, \{\mathtt{u_j}\}, \Psi, \sigma_\mathtt{i})$ on the blockchain $\mathcal{B}$. $\mathcal{B}$ validates the Ψ and updates the on-chain balances. If this operation is successful, $\mathcal{B}$ returns `success` with $\top$ to u_i; otherwise, it returns `fail` with $\perp$. If any of the neighbors u_j, does not agree with the updated balances, the blockchain can be used to resolve the conflict by providing the transaction state Ψ_{ij}.

5 Security Analysis

Theorem 1. *FlexiPCN protocol UC-realizes the ideal functionality* $\mathcal{F}_{\mathtt{FlexiPCN}}$: $(\mathcal{F}_{\mathtt{ECDSA}}, \mathcal{F}_{\mathcal{B}}, \mathcal{F}_{\mathcal{C}}, \mathcal{F}_{\mathtt{anon}})$*-hybrid model, provided that the digital signature scheme is existentially unforgeable.*

Proof. The `OpenChannel` and `CloseChannel` operations are discussed in [5], and `Payment` operation is also discussed in [23]; so here we are focusing on the remaining operations: `CoinDeposit`, `CoinAllocation`, `CoinShift`, and `CoinSettlement`. This simulator $\mathcal{S}$ uses a secure cryptographic primitive called Elliptic Curve Digital Signature Algorithm (ECDSA), which is assumed to be secure.

$\underline{\texttt{CoinDeposit}}$

$\underline{u_i \textbf{ is honest:}}$ When u_i sends $(\underline{\texttt{CoinDeposit}}, \mathtt{u_i}, \{\mathtt{u_j}\}, \omega_\mathtt{i}) \hookrightarrow \mathcal{F}$, generate signature σ_i of user u_i on message $(\{u_j\}, \omega_i)$ where $\{u_j\}$ is the set of active adjacent channel user, ω_i is the amount to be deposited or locked in the contract functionality $\mathcal{C}$, and sends $(\underline{\texttt{CoinDeposit}}, \mathtt{u_i}, \{\mathtt{u_j}\}, \omega_\mathtt{i}, \sigma_\mathtt{i})$ to $\mathcal{C}$ on behalf of user u_i. $\mathcal{C}$ returns $(\texttt{deposit} - \texttt{confirm})$ to $\mathcal{F}$, and $\mathcal{F}$ returns $\top$ to u_i.

$\underline{u_i \textbf{ is corrupt:}}$ When u_i sends $(\underline{\texttt{CoinDeposit}}, \mathtt{u_i}, \{\mathtt{u_j}\}, \omega_\mathtt{i}, \sigma_\mathtt{i}) \hookrightarrow \mathcal{C}$, $\mathcal{C}$ sends $(\underline{\texttt{CoinDeposit}}, \mathtt{u_i}, \{\mathtt{u_j}\}, \omega_\mathtt{i}, \sigma_\mathtt{i}) \hookrightarrow \mathcal{F}$ on behalf of u_i. $\mathcal{F}$ returns $\bot$ to u_i.

$\underline{\texttt{CoinAllocation :}}$

$\underline{u_i \textbf{ is honest:}}$ When u_i sends $(\underline{\texttt{CoinAllocation}}, \mathtt{u_i}, \{\mathtt{u_j}\}, \omega_\mathtt{i}) \hookrightarrow \mathcal{F}$, generate signature $\sigma_i{}'$ of u_i on $(u_i, \{u_j\}, \omega_i)$, sends $(\underline{\texttt{CoinAllocation}}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \frac{\omega_\mathtt{i}}{|\{\mathtt{u_j}\}|}, \sigma_\mathtt{ij}{}')$ to each adjacent user $u_j \in \{u_j\}$ on behalf of u_i.

For each $u_j \in \{u_j\}$:

- If u_j is corrupt: It sends $(\texttt{confirm} - \texttt{allocate}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \sigma'_\mathtt{ji})$ to u_i where σ'_{ji} is the signature of u_j on $(c_{\langle u_i, u_j \rangle}, \frac{\omega_i}{|\{u_j\}|})$, send $(\underline{\texttt{CoinAllocation}}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \frac{\omega_\mathtt{i}}{|\{\mathtt{u_j}\}|})$ $\hookrightarrow \mathcal{F}$ on behalf of u_j.
- If u_j is honest: It sends $(\texttt{confirm} - \texttt{allocate}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}) \hookrightarrow \mathcal{F}$, generate signature σ'_{ji} of u_j on $(c_{\langle u_i, u_j \rangle}, \frac{\omega_i}{|\{u_j\}|})$, and sent it to u_i on behalf of u_j.

$\underline{u_i \textbf{ is corrupt:}}$ When u_i sends $(\underline{\texttt{CoinAllocation}}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \frac{\omega_\mathtt{i}}{|\{\mathtt{u_j}\}|}, \sigma'_\mathtt{ij})$ to u_j, where $u_j \in \{u_j\}$ and u_j is honest, send $(\underline{\texttt{CoinAllocation}}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \frac{\omega_\mathtt{i}}{|\{\mathtt{u_j}\}|}) \hookrightarrow \mathcal{F}$ on behalf of u_i and send $(\texttt{allocate} - \texttt{request}, \mathtt{u_j})$ to $\mathcal{F}$.

For each $u_j \in \{u_j\}$:

- If u_j is corrupt: It sends $(\texttt{confirm} - \texttt{allocate}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle})$ to $\mathcal{F}$ on behalf of u_j.
- If u_j is honest: It sends $(\texttt{confirm} - \texttt{allocate}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}) \hookrightarrow \mathcal{F}$, generate signature of u_j on $(c_{\langle u_i, u_j \rangle}, \frac{\omega_i}{|\{u_j\}|})$ and send $(\underline{\texttt{CoinAllocation}}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \frac{\omega_\mathtt{i}}{|\{\mathtt{u_j}\}|}, \sigma'_\mathtt{ij})$ to u_i on behalf of u_i.

$\underline{\texttt{CoinShift :}}$

$\underline{u_i \textbf{ is honest:}}$ When u_i sends $(\underline{\texttt{CoinShift}}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_k} \rangle}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \nu_\mathtt{ik}) \hookrightarrow \mathcal{F}$, send $(\underline{\texttt{CoinShift}}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_k} \rangle}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \nu_\mathtt{ik})$ to u_k on behalf of u_i.

- If u_k is honest: It generates signature σ''_{ki} on $(c_{\langle u_i, u_k \rangle}, \varUpsilon''_{ik}, \mathtt{C}''_{ik})$ and send $(c_{\langle u_i, u_k \rangle}, \varUpsilon''_{ik}, \mathtt{C}''_{ik}, \sigma''_{ki}) \hookrightarrow \mathcal{F}$. $\mathcal{F}$ sends $(c_{\langle u_i, u_k \rangle}, \varUpsilon''_{ik}, \mathtt{C}''_{ik}, \sigma''_{ki})$ to u_i on behalf of u_k. u_i sends $(\mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_k} \rangle}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \Upsilon''_\mathtt{ij}, \mathtt{C}''_\mathtt{ij}) \hookrightarrow \mathcal{F}$. $\mathcal{F}$ generates signatures σ''_{ik} on $(c_{\langle u_i, u_k \rangle}, \varUpsilon''_{ik}, \mathtt{C}''_{ik})$ and σ''_{ij} on $(c_{\langle u_i, u_j \rangle}, \varUpsilon''_{ij}, \mathtt{C}''_{ij})$, send $(\mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_k} \rangle}, \mathtt{c}_{\langle \mathtt{u_i}, \mathtt{u_j} \rangle}, \Upsilon''_\mathtt{ij}, \mathtt{C}''_\mathtt{ij}, \langle \sigma''_{ij}, \sigma''_{ik}, \sigma''_{ki} \rangle)$ to u_j on behalf of u_i.

  - If u_j is honest: It sends $(\mathtt{coin-shift-ok}) \hookrightarrow \mathcal{F}$, generate signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$, send $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k on behalf of u_i.
  - If u_j is corrupt: It sends signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$ to u_i, sends $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k on behalf of u_i.

- If u_k is corrupt: It generates signature $\sigma_{ki}^{''}$ on $(c_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''})$ and sends $(c_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \sigma_{ki}^{''})$ to u_i. u_i sends $(c_{\langle u_i,u_k\rangle}, c_{\langle u_i,u_j\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''}, \sigma_{ki}^{''}) \hookrightarrow \mathcal{F}$, generates signatures $\sigma_{ik}^{''}$ on $(c_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''})$ and $\sigma_{ij}^{''}$ on $(c_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$, send $(\mathtt{c}_{\langle u_i,u_k\rangle}, \mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''}, \langle\sigma_{ij}^{''}, \sigma_{ik}^{''}, \sigma_{ki}^{''}\rangle)$ to u_j on behalf of u_i.
  - If u_j is honest: It sends $(\mathtt{coin-shift-ok}) \hookrightarrow \mathcal{F}$, generate signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$, send $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k on behalf of u_i.
  - If u_j is corrupt: It sends signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$ to u_i, sends $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k on behalf of u_i.

u_i is corrupt: When u_i sends $(\underline{\mathtt{CoinShift}}, \mathtt{c}_{\langle u_i,u_k\rangle}, \mathtt{c}_{\langle u_i,u_j\rangle}, \nu_{\mathtt{ik}}) \hookrightarrow \mathtt{u_k}$.

- If u_k is honest: It generates signature $\sigma_{ki}^{''}$ on $(c_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''})$ and send $(c_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''}, \sigma_{ki}^{''}) \hookrightarrow \mathcal{F}$. $\mathcal{F}$ sends $(\mathtt{c}_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''}, \sigma_{ki}^{''})$ to u_i on behalf of u_k. u_i generates signatures $\sigma_{ik}^{''}$ on $(\mathtt{c}_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''})$ and $\sigma_{ij}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$, and send $(\mathtt{c}_{\langle u_i,u_k\rangle}, \mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''}, \langle\sigma_{ij}^{''}, \sigma_{ik}^{''}, \sigma_{ki}^{''}\rangle)$ to u_j.
  - If u_j is honest: It sends $(\mathtt{coin-shift-ok}) \hookrightarrow \mathcal{F}$. u_i generate signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$, send $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k.
  - If u_j is corrupt: It generates signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_k\rangle}, \mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$ and sends $(\mathtt{c}_{\langle u_i,u_j\rangle}, \sigma_{ij}^{''})$ to u_i, it sends $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k.

- If u_k is corrupt: It generates signature $\sigma_{ki}^{''}$ on $(\mathtt{c}_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''})$ and sends $(c_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''}, \sigma_{ki}^{''})$ to u_i. It generates signatures $\sigma_{ik}^{''}$ on $(\mathtt{c}_{\langle u_i,u_k\rangle}, \Upsilon_{ik}^{''}, \mathtt{C}_{ik}^{''})$ and $\sigma_{ij}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$, and sends $(\mathtt{c}_{\langle u_i,u_k\rangle}, \mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''}, \langle\sigma_{ij}^{''}, \sigma_{ik}^{''}, \sigma_{ki}^{''}\rangle)$ to u_j.
  - If u_j is honest: It sends $(\mathtt{coin-shift-ok}) \hookrightarrow \mathcal{F}$, generate signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$, send $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k.
  - If u_j is corrupt: It generates signature $\sigma_{ji}^{''}$ on $(\mathtt{c}_{\langle u_i,u_k\rangle}, \mathtt{c}_{\langle u_i,u_j\rangle}, \Upsilon_{ij}^{''}, \mathtt{C}_{ij}^{''})$ and sends $(\mathtt{c}_{\langle u_i,u_j\rangle}, \sigma_{ij}^{''})$ to u_i, it sends $(\mathtt{c}_{\langle u_i,u_j\rangle}, \langle\sigma_{ik}^{''}, \sigma_{ij}^{''}, \sigma_{ji}^{''}\rangle)$ to u_k.

$\underline{\mathtt{CoinSettlement:}}$

u_i is honest: When u_i sends $(\underline{\mathtt{CoinSettlement}}, \mathtt{u_i}, \Psi) \hookrightarrow \mathcal{F}$ where Ψ is the most recent updated state of the payment channels, generate signature σ_i of user u_i on Ψ, and sends $(\underline{\mathtt{CoinSettlement}}, \mathtt{u_i}, \{\mathtt{u_j}\}, \omega_{\mathtt{i}}, \sigma_{\mathtt{i}})$ to $\mathcal{C}$ on behalf of user u_i. If the pending payment requests are completed and updated channel balances, $\mathcal{C}$ sends confirmation $(\mathtt{settlement-confirm})$ to $\mathcal{F}$, and $\mathcal{F}$ returns $\top$ to u_i.

u_i is corrupt: When u_i sends $(\underline{\mathtt{CoinSettlement}}, \mathtt{u_i}, \Psi, \sigma_{\mathtt{i}}) \hookrightarrow \mathcal{C}$, $\mathcal{C}$ sends $(\underline{\mathtt{CoinSettlement}}, \mathtt{u_i}, \{\mathtt{u_j}\}, \Psi, \sigma_{\mathtt{i}}) \hookrightarrow \mathcal{F}$ on behalf of u_i. $\mathcal{F}$ returns $\perp$ to u_i.

6 Conclusion

In this paper, we have proposed a novel flexible payment channel network called FlexiPCN, which rebalances the channel completely off-chain. In FlexiPCN, coins can be deposited per user instead of per channel, allowing users to move coins between channels without going on-chain or setting up a cycle. We have formalized and studied the security of the FlexiPCN protocol using the Universal Composability (UC) framework. The implementation of FlexiPCN is in progress, and we are looking at various applications for it.

Acknowledgement

We acknowledge the Ministry of Education (MoE), Government of India for providing fellowship under Ph.D. program to complete this work.

References

1. Raiden Network: Fast, cheap, scalable token transfers for Ethereum. `https://raiden.network/` (2018)
2. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
3. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Gün Sirer, E., et al.: On scaling decentralized blockchains: (a position paper). In: Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20. pp. 106–125. Springer (2016)
4. Dotan, M., Pignolet, Y.A., Schmid, S., Tochner, S., Zohar, A.: Sok: cryptocurrency networking context, state-of-the-art, challenges. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–13 (2020)
5. Ge, Z., Zhang, Y., Long, Y., Gu, D.: Shaduf: Non-cycle payment channel rebalancing. In: 29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022
6. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: International Conference on Financial Cryptography and Data Security. pp. 201–226. Springer (2020)
7. Harris, J., Zohar, A.: Flood & loot: A systemic attack on the lightning network. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 202–213 (2020)
8. Herrera-Joancomartí, J., Navarro-Arribas, G., Ranchal-Pedrosa, A., Pérez-Solà, C., Garcia-Alfaro, J.: On the difficulty of hiding the balance of lightning network channels. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security. pp. 602–612 (2019)
9. Hong, Z., Guo, S., Zhang, R., Li, P., Zhan, Y., Chen, W.: Cycle: Sustainable off-chain payment channel network with asynchronous rebalancing. In: 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 41–53. IEEE (2022)

10. Khalil, R., Gervais, A.: Revive: Rebalancing off-blockchain payment networks. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS, Dallas, TX, USA, October 30 - November 03, 2017. pp. 439–453
11. Li, P., Miyazaki, T., Zhou, W.: Secure balance planning of off-blockchain payment channel networks. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications. pp. 1728–1737. IEEE (2020)
12. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 455–471 (2017)
13. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019
14. Mavroudis, V., Wüst, K., Dhar, A., Kostiainen, K., Capkun, S.: Snappy: Fast on-chain payments with practical collaterals. arXiv preprint arXiv:2001.01278 (2020)
15. Mizrahi, A., Zohar, A.: Congestion attacks in payment channel networks. In: International Conference on Financial Cryptography and Data Security. pp. 170–188. Springer (2021)
16. Mohanty, S.K., Tripathy, S.: n-htlc: Neo hashed time-lock commitment to defend against wormhole attack in payment channel networks. Computers & Security **106**, 102291 (2021)
17. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf
18. Pérez-Solà, C., Ranchal-Pedrosa, A., Herrera-Joancomartí, J., Navarro-Arribas, G., Garcia-Alfaro, J.: Lockdown: Balance availability attack against lightning network channels. In: International Conference on Financial Cryptography and Data Security. pp. 245–263. Springer (2020)
19. Poon, J., Dryja, T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. `https://lightning.network/lightning-network-paper.pdf` (2016)
20. Rohrer, E., Malliaris, J., Tschorsch, F.: Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 347–356. IEEE (2019)
21. Rohrer, E., Tschorsch, F.: Counting down thunder: Timing attacks on privacy in payment channel networks. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 214–227 (2020)
22. Szabo, N.: The idea of smart contracts. Nick Szabo's papers and concise tutorials **6**(1), 199 (1997)
23. Tripathy, S., Mohanty, S.K.: Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In: International Conference on Financial Cryptography and Data Security. pp. 481–495. Springer (2020)
24. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger, https://ethereum.github.io/yellowpaper/paper.pdf