NOCUST – A Non-Custodial 2$^{\text{nd}}$-Layer Financial Intermediary

Rami Khalil
Liquidity.Network
rami@liquidity.network

Arthur Gervais
Liquidity.Network
arthur@liquidity.network

ABSTRACT

Bitcoin is meant to offer a payment system where the users are custodians of their funds instead of entrusting a trusted financial institution. The limited transaction throughput of such permissionless blockchains, however, results for example in volatile transaction prices that hardly fit into traditional service level agreements required by professional institutions and cannot accommodate micro-transactions.

We present a novel non-custodial 2$^{\text{nd}}$-layer financial intermediary solution secure against double-spending that guarantees users control of funds through leveraging a smart contract enabled decentralized blockchain ledger as a means of dispute resolution. Two-party payment channels networks have been proposed as building blocks for trust-free payments that do not exhaust the resources of the blockchain; however, they bear multiple fundamental limitations. NOCUST is a specification for secure N-party payment hubs with improved transaction utility, cheaper operational costs and leaner user enrollment.

1 INTRODUCTION

Since the beginning of centralized banking in Mesopotamia [1], finance intermediaries evolved as middlemen between, e.g. parties that have surplus capital and others that desire access to liquid funds. Such finance intermediaries traditionally act as custodians, i.e., they (temporarily) own the transmitted funds, and therefore are entrusted with correct monetary policy.

While the emergence of decentralized ledgers such as Bitcoin has portrayed a mechanism of performing financial transactions without a centralized intermediary, low-throughput and privacy constraints are fundamentally hindering the practical use of such ledgers. In particular the volatility of transaction fees do not align with the business needs for Service Level Agreements (SLA), such as a guaranteed transaction throughput and high availability of the financial intermediary, while small value transfers (microtransactions) are impractically expensive.

To improve transaction utility, different classes of blockchain scaling solutions are being pursued. Alternative consensus mechanisms [2–5] typically introduce different trust assumptions, sharding [6; 7] attempts to partition the network into smaller shards that reach faster consensus, while second layer payment channel scaling solutions [8–11] reduce the load on the blockchain ledger by performing operations off-chain securely.

Two-party payment channels establish a direct peer-to-peer payment medium between two parties, where individual transactions on their privately maintained two-party ledger are not written to the blockchain, while, at any given time, both parties are guaranteed to be able to claim their legitimate funds in the global blockchain. Linked payments across a chain of payment channels are transmitted across two-party payment channel networks [8; 10] and allow performing payments between parties that are not directly connected by a payment channel. Numerous contributions address the performance characteristics of payment networks [9; 11–13].

However, two-party payment channel networks face multiple fundamental flaws: (i) two-party payment channel based hubs would require insurmountable amounts of frozen collateral, (ii) channel establishment requires an expensive on-chain transaction, (iii) two-party payment channels rely on complex routing topologies which need to be setup and maintained, (iv) funds allocated to a payment channel are typically bound between two parties and can only be transferred further over fee-contingent routes, (v) current payment channels require always-online observers to deter potential misbehaviour of the involved parties.

This work. In this work, we propose a novel off-chain payment hub construction named NOCUST, which allows the operation of a non-custodial finance intermediary, that by design can achieve the same transaction throughput as traditional custodians. A user can open a payment channel directly with a NOCUST hub off-chain, foregoing the need for a costly on-chain channel initialization transaction. NOCUST allows a set of participants to securely transact over a single payment hub — their allocated funds can therefore be used freely among the members of the hub. NOCUST's structure alleviates the burden on the off-chain network to route payments, while multiple NOCUST hubs can be interconnected e.g. over traditional two-party payment channels with fairly static and long-lived peering agreements.

Defining transaction finality as the point at which a transfer is irreversible on-chain, we demonstrate how NOCUST achieves delayed finality after a disputable time window, and show how the hub operator can choose to stake an amount of collateral anywhere between zero and the transaction volume during the disputable time window as a trust model parameter that completely eliminates ephemeral counter-party risk if desired. We argue that for many real-world applications, a certain degree of reliability on the system of the hub operator is sensible, because the hub's smart contract would seize to function if its integrity is not provable, effectively ruining the hub operator's business.

Contrary to traditional payment channel networks, we show how a NOCUST payment hub can easily manage its collateral in bulk, significantly reducing the operating costs of hub payment channels. The main contributions of our work are as follows:

- We provide a novel payment hub construction NOCUST for off blockchain payments that provides N-party off-chain payment channels where we introduce a novel commitment scheme to account for user's balances.
- Depending on how trustworthy the payment hub is, the hub either requires no additional locked up funds per additional user or a progressive amount of collateral, capped at the respective transaction volume of a disputable time

window. We show how this collateral can be managed efficiently in bulk, as opposed to traditional payment channels.
- Allocated funds of a user within a payment hub can be used to pay any other member of the payment hub. This significantly reduces the routing and network complexity of existing payment channel designs.
- We show how users of a payment hub can securely maintain custody of their funds, even in the absence of the hub's availability or under its adversarial behavior.

The remainder of the paper is organized as follows. In Section 2, we provide the necessary background and related work overview on permissionless blockchains and payment channel networks. In Section 3 we present the NOCUST architecture, while we analyze its security and privacy in Section 4. We evaluate NOCUST in terms of its usability and practicality in Section 5. In Section 6 we outline possible avenues for future work on NOCUST, and lastly, we conclude the paper in Section 7.

2 BACKGROUND AND RELATED WORK

In this section, we provide the necessary background on permissionless blockchains such as Bitcoin and Ethereum, and discuss existing payment channel networks.

2.1 Decentralized Ledgers

Bitcoin [14] allows mutually mistrusting peers to trade, without relying on a centralized trusted third party. Inspired by Bitcoin, other blockchains surfaced, e.g., Ethereum [15] which extends Bitcoin's transaction language to a Turing complete programming language to ease the development of so-called smart contracts. We refer the reader to related work [16] for more in-depth background on decentralized ledgers.

The blockchain's primary intention is to provide a time stamping service that can act as an electronic payment solution which solves the double-spending problem. That is the problem of spending an electronic coin multiple times. The majority of the existing blockchains rely on a so-called *Proof of Work* (PoW) [17; 18], which is a computationally expensive puzzle and allows for a permissionless blockchain operation - i.e. any peer can join and leave the network at any time.

The central costs associated to permissionless blockchains stems from the requirement that all peers are required to be made aware of all transactions in the network to not be vulnerable to double-spending. Bitcoin currently only supports up to about 10 transactions per second [19]. This throughput is unlikely to grow for a single consolidated network beyond 100 transactions per second (assuming the same underlying Internet topology) with simple re-parametrization [20].

2.2 Two-party Payment Channels

Two-party payment channels establish direct peer-to-peer payment channels between two parties. A payment channel can be seen as a private two-party ledger, which is instantiated and closed with a respective on-chain transaction. During the channel's lifetime, the channel is privately maintained and does not require ongoing communication with the underlying blockchain. The security of payment channel payments is guaranteed by the amount of escrow that the channel holds on-chain and it's ability to recourse to the blockchain in case of disputes. This on-chain escrow which is gradually used off-chain makes sure that the participants are only allowed to spend their rightful amounts.

Because a payment channel transaction avoids costly on-chain transactions (besides channel establishment and closure), the direct service of miners is not required and the transaction costs are significantly reduced. As such payment channels (re)enable the use of micro transactions on the blockchain. The transaction rate is primarily limited by the network bandwidth between the participating peers and the respective channel collateral.

For a pair of individuals that are not directly connected via a payment channel, a payment can be routed along a set of payment channels, i.e. over a payment channel network. This avoids to open individual payment channels with each counter-party that one might interact with. Such payment networks are envisioned to enhance the usability and practicality of payment channels.

Routing payments over a payment network has certain analogies to Internet packet routing, with additional routing restrictions. Intermediate hops on a routing path are required to offer sufficient collateral along the payment path, and if one intermediate payment were to fail, the other intermediate payments are also invalidated. Linked payments are therefore atomically executed or invalidated. Intermediate hops are eligible to collect payment forwarding fees.

Several off-chain payment solutions have been proposed and can be divided into two categories. The first category relies on blockchain based *time locks* (e.g. by Decker *et al.* [21]). The channel starts with a commitment transaction which for example lasts for 10 days. The subsequent commitment transaction will then last 9 days, and can thus be spent before the first transaction. The second category of payment channels relies on punishment, i.e. if one party misbehaves, the other party can claim all funds of the channel. One instance of this payment channel is the Lightning Network [8]. The Lightning Network relies on Bitcoin, while the Raiden Network [10] is currently in development for the Ethereum blockchain. Existing payment channels are still in early development and therefore allow for several improvement proposals. Sprites [9], inspired by Lightning and Raiden aims to minimize the worst-case collateral costs of indirect off-chain payments. Flare [11] is another proposal to optimize the search process of finding a payment route. Bolt [22] provides different constructions that allow for privacy preserving off-chain payment channels. BitcoinJ, a lightweight Bitcoin client implementation, also supports micropayment channels [23]. The Orchid Network proposes probabilistic micropayments, where the recipient does not necessarily need to have deposited funds in escrow [24]. Probabilistic payments, however are only more efficient than payment channels whenever the service provided is continuous and granular enough for the probabilistic variance to become negligible.

2.3 Alternatives

TumbleBit [25] is an anonymous payment protocol, fully compatible with nowadays Bitcoin protocol and allows parties to make payments through an untrusted Tumbler (which can operate as hub as well). TumbleBit claims to provide privacy properties such that no-one, not even the Tumbler, can tell which payer paid which payee during a TumbleBit epoch. Nothing apparently, however, prevents the hub to deny payments to individual users, finally enabling the hub to reject all but the victims and hub controlled payments. The TumbleBit hub would, therefore, be able to infer who got paid through the hub. TumbleBit requires that the hub operator opens a directed channel with each recipient, where the full collateral of the allowed payments is to be deposited. This significantly complicates the operations of the payment hub because collateral for each channel needs to be managed independently. Our proposed payment hub allows collateral to be managed in bulk in a centrally managed smart contract, facilitating the operations of the hub and allowing a gradual collateral attribution.

Perun [26] is an off-chain protocol to establish "virtual" two-party channels on top of two-party channels established on-chain, referred to as "ledger" channels in the work. The motivation is to enable two parties which are unconnected by an on-chain payment channel to establish an off-chain connection through leveraging a pre-existing channel. The work describes how an intermediary with multiple on-chain connections may facilitate these virtual channel establishments to enable payments between two of its unconnected peers. The security guarantees provided by Perun do not prevent a client and an intermediary from successfully double spending the balance of a ledger channel in a set of virtual channels off-chain, and finalizing the expenditure on-chain. Instead, the security guarantees provide provable misbehavior in the worst case through non-repudiable signed messages. Moreover, the collateral of the intermediary is fragmented across its channels. In contrast, our hub construct leverages provable integrity to prevent a malicious intermediary from double-spending its balance and being able to finalize the double spent transactions on-chain.

Plasma [27] is a specification for connecting a UTXO ledger with a parent account-based ledger, where minting and reclaiming unspent transaction may only take place on the parent ledger. Parties transact through authorizing spends of their UTXOs towards the intended recipients, as in Bitcoin, and sending their authorizations to the UTXO ledger network, which then aggregates the transaction set into blocks and commits to them on the parent network. With the incorporation of a UTXO model comes all of the inefficiencies of transaction history validation and the inflexibility of transaction output expenditure, and in the absence of a consensus mechanism full block validation is required by clients. As a participant holds more transaction outputs within the child ledger, their costs for securely using their accounts increase over time in terms of validation effort and dispute resolution. Moreover, plasma does not specify a mechanism for mitigating risk of reversal on yet to be finalized transactions, parting its guarantees from those of payment channel networks. In our work, we design a more efficient bi-modal ledger structure that mitigates these issues, permitting clients to more efficiently verify the integrity of their accounts and requiring less on-chain data for dispute.

2.4 Fundamental drawbacks of existing designs

In the following we elaborate on the fundamental drawbacks of existing payment channel designs.

2.4.1 Fragmented collateral. Traditional payment channels require collateral to be locked up for every channel. If one were to construct a payment hub with 2-party payment channels, the hub operator would be required to lock up a substantial amount of collateral. Given for example a hub with 1 Million users, and each user receiving on average 10'000 USD of transaction volume over a period of 1 month, this would require the hub operator to lock up a total of 10 Billion USD. This amount could be reduced by dynamically adjusting the respective channels of the users, which however would require costly on-chain transactions. It's therefore likely that most of the locked up collateral would not be used frequently.

2.4.2 Expensive Channel Setup. Continuing on the example given in the previous paragraph, a hub operator with 1 Million users would be required to setup 1 Million on-chain transactions for each channel setup. This in itself represents a substantial investment (beyond 1'000'000 USD on Ethereum given the current gas prices).

2.4.3 Costly routing. Locked up funds in traditional payment channels cannot be used directly with other nodes in the network. Routing payments certainly alleviates this issue, but still limits the available funds to a particular set of routes.

Route finding itself, has shown to be a significant difficulty in the realization of two-party payment channel networks. Peers might become unresponsive, which requires payments to be diverted over newer, possibly more costly routes.

2.4.4 Online Watchdog requirements. The two parties of a traditional payment channel should remain online to observe their channel state continuously. That is, because if one party were to transition offline, the other party could attempt to close the channel with an outdated state and effectively double-spend a disputable amount. To alleviate this online presence requirement, users could outsource the so called watchdog duty to a third party, which however introduces new trust assumptions along with additional costs.

2.4.5 Reduced Privacy. Because off-chain transactions are no longer recorded in the readable blockchain, one would argue that payment channel networks offer better privacy guarantees than on-chain transactions. Related work however has argued otherwise and proposed privacy enhanced payment channel designs [28–31].

2.4.6 Double-Spending Attacks on Blockchain Congestion. Under a congested blockchain, channel termination could result in a bidding war between a set of participants in a payment channel. Incorrect channel termination can in particular be aggravated in the event of a *mass-exit*, where many participants wish to close their payment channels, and therefore trigger an aggravated blockchain congestion. Note that the respective dispute resolution mechanism might in some cases not be worth considering if the disputed value is insignificant.

3 NOCUST ARCHITECTURE

In the this section, we present the non-custodial intermediary construction NOCUST, denoted as $\not\subset$ for brevity.

3.1 Prerequisite System Model

In the following we outline the considered system model. A $\not\subset$ instance is designed to operate atop a blockchain layer supporting smart contracts while its clients communicate directly off-chain.

In our scheme, the blockchain $\mathcal{BC}$ is considered as an integrity protected and immutable root of trust that comprises a decentralized database containing a global view of accounts, their balances and transactions, and extra associated data. Each account in the ledger is controlled by its own private key, that only the owner of the account should know. A transaction from any account cannot be authorized without possession of its respective private key. Authorized modifications to the ledger are considered to be globally available after a block is generated, on average every predetermined block time T. Due to the characteristic of providing an average block time T, a blockchain can also be viewed as a timestamping service. In the following sections we refer to the average time for block generation as an *era*.

In addition to primitive ledger transactions that transfer balance from one account to another, our scheme also requires a smart contract execution environment, such that provided by Ethereum [15]. It's important to note that Ethereum's smart contracts are allowed to hold a balance in the ledger, and control it according to their programming. We assume that once a smart contract is published, it cannot be modified, nor can a result outside the bounds of its correct execution be accepted on the decentralized ledger $\mathcal{BC}$.

We also assume an underlying communication network, where all the participants can communicate directly off-chain (e.g. via TCP connections).

3.2 Overview

The intermediary $\not\subset$ is composed of two fundamental building blocks: (i) an honest verifier smart contract (denoted as $\mathcal{V}_{\not\subset}$) and (ii) an operator server (denoted as $O_{\not\subset}$). $\mathcal{V}_{\not\subset}$ can operate on any Turing-complete enabled blockchain $\mathcal{BC}$, while $O_{\not\subset}$ can be embodied as an internet-reachable interactive server.

Clients hold their private keys that control their identities and use them to sign off-chain messages while communicating with $O_{\not\subset}$. They also use the private keys to control their on-chain wallets and interact with $\mathcal{V}_{\not\subset}$.

$O_{\not\subset}$ posses its own private key for its identity, which can perform some privileged operations in $\mathcal{V}_{\not\subset}$. $O_{\not\subset}$ periodically commits to the off-chain ledger state with $\mathcal{V}_{\not\subset}$ as described in Section 3.4.1.

3.2.1 Example Payment Scenario. Bob, a user of a payment hub, would carry out the following steps to transfer an off-chain payment to another user, Alice. We assume that both Bob and Alice own at least 1 unit of cryptocurrency (e.g. ether) on the blockchain $\mathcal{BC}$.

(1) Bob and Alice choose to transact through $\not\subset$.
(2) Alice deposits 1 Ether in the smart contract $\mathcal{V}_{\not\subset}$. Note that for reception only, Alice is not required to deposit funds into $\mathcal{V}_{\not\subset}$.
(3) Bob deposits 1 Ether in the smart contract $\mathcal{V}_{\not\subset}$.
(4) Bob signs an `IOU` designated to Alice for 0.1 Ether and sends the `IOU` to $O_{\not\subset}$.
(5) $O_{\not\subset}$ then notifies Alice of the `IOU` and awaits a signed receipt from Alice.
(6) Alice signs a receipt of the `IOU` and sends it to $O_{\not\subset}$.
(7) $O_{\not\subset}$ confirms the validity of the `IOU` execution, ratifies it, then sends its signatures on the `IOU` to Alice and Bob.
(8) $O_{\not\subset}$ synchronizes with $\mathcal{V}_{\not\subset}$.
(9) Alice may withdraw up to 1.1 Ether from $\mathcal{V}_{\not\subset}$.
(10) Bob may withdraw up to 0.9 Ether from $\mathcal{V}_{\not\subset}$.

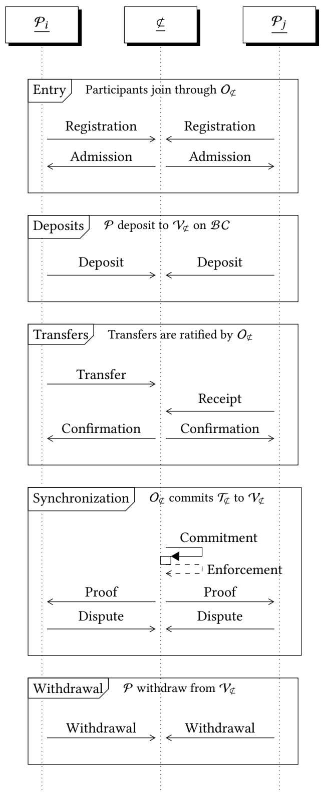

$\mathcal{P}_i$
$\not\subset$
$\mathcal{P}_j$
Entry
Participants join through $O_{\not\subset}$
Registration
Registration
Admission
Admission
Deposits
$\mathcal{P}$ deposit to $\mathcal{V}_{\not\subset}$ on $\mathcal{BC}$
Deposit
Deposit
Transfers
Transfers are ratified by $O_{\not\subset}$
Transfer
Receipt
Confirmation
Confirmation
Synchronization
$O_{\not\subset}$ commits $\mathcal{T}_{\not\subset}$ to $\mathcal{V}_{\not\subset}$
Commitment
Enforcement
Proof
Proof
Dispute
Dispute
Withdrawal
$\mathcal{P}$ withdraw from $\mathcal{V}_{\not\subset}$
Withdrawal
Withdrawal

Figure 1: A sequential view of a NOCUST instance life-cycle. In practicality, transfers, deposits, and withdrawals may interleave post entry. Receipt of an off-chain transfer is possible after admission and does not require a prior deposit.

3.2.2 Operational Requirements. To disqualify $\not\subset$ from being a trusted custodian of its user's funds and the payments it facilitates, it would have to provide the following guarantees:

- Funds may not be transacted without user authorization.
- Users must always be able to withdraw their off-chain balance to their on-chain accounts.

These guarantees need to be provided by the system to an honest participant $\mathcal{P}_i$ regardless of the behavior of $\not\subset$ or of any other participant $\mathcal{P}_j$ $(i \neq j)$. We prove these guarantees in Section 4.

Moreover, to qualify as an off-chain solution, rather than a sidechain, there needs to be no reliance on a full consensus mechanism in the second layer that demands mining or full inspection of the contents of the entire off-chain ledger by all participants.

In NOCUST we define how a participant $\mathcal{P}_i$ could interact with an intermediary $\not\subset$ to enact off-chain transactions while satisfying the above requirements through utilizing the following components:

- A bimodal specialized balance ledger $\mathcal{B}$.
- A specification for $\mathcal{V}_{¢}$ to manage $\mathcal{B}^G$.
- A specification for $O_{¢}$ to manage $\mathcal{B}^L$.
- A specification for interactions between $\mathcal{P}$, $O_{¢}$, and $\mathcal{V}_{¢}$.

3.3 $\mathcal{B}$ Bimodal Ledger

In this section we present a simple scheme for managing local and global balance and transaction information in $\mathcal{B}$ such that it can be efficiently utilized to provide the secure operation of a ¢ instance.

3.3.1 Separation. The bimodal $\mathcal{B}$ is separated into the locally stored "off-chain" $\mathcal{B}^L$, which contains information related to balances and transfers performed through $O_{¢}$, and the globally stored "on-chain" $\mathcal{B}^G$, which comprises information on balances and operations performed through $\mathcal{V}_{¢}$. It is important to note that different parties may have different views of the contents of $\mathcal{B}^L$, but contents of $\mathcal{B}^G$ are assumed to be globally consistent.

3.3.2 Time Progression. The information in $\mathcal{B}^L$ is committed to on $\mathcal{B}^G$ to make the transfers carried out through $O_{¢}$ enforceable by $\mathcal{V}_{¢}$ (ie. allow a $\mathcal{P}_i$ to withdraw funds received through $O_{¢}$ from a $\mathcal{P}_j$ via $\mathcal{V}_{¢}$). This synchronization commitment is sent periodically from $O_{¢}$ to $\mathcal{V}_{¢}$, and the duration of this period is referred to as an *eon*, where the synchronization occurs at most once per *eon*. An *eon*, within the context of our scheme, is further divided into a fixed number of *eras*. Within the context of a blockchain ecosystem, an *era* represents the amount of time taken to generate one block $\mathcal{BC}$, and thus commit a set of modifications to $\mathcal{B}^G$. Consequently, an *eon* represents the amount of time for a fixed number of blocks to be generated. We also use the term *epoch* to denote a quarter of an *eon*. We use $\mathcal{B}(e)$ to refer to the state of the balance ledger at *eon* number *e as of all eras passed*, and similarly for $\mathcal{B}^L(e)$ and $\mathcal{B}^G(e)$.

3.3.3 Locally Stored Information. For every *eon e*, for every $\mathcal{P}_i$, $\mathcal{B}^L(e)$ can store the following entries:

- $A_i(e)$: Initially allotted balance of $\mathcal{P}_i$ for e.
- $R_i(e)$: Total amount received off-chain by $\mathcal{P}_i$ during e.
- $S_i(e)$: Total amount sent off-chain by $\mathcal{P}_i$ during e.
- $T_i(e)$: The set of off-chain transactions sent or received by $\mathcal{P}_i$ during e.

3.3.4 Globally Stored Information. For every *eon e*, for every $\mathcal{P}_i$, $\mathcal{B}^G(e)$ can store the following entries:

- $\mathcal{D}_i(e)$: Total amount deposited by $\mathcal{P}_i$ during e.
- $\mathcal{W}_i(e)$: Total amount requested for withdrawal by $\mathcal{P}_i$ during e.
- $\mathcal{X}_i^b(e)$: A challenge by $\mathcal{P}_i$ against the integrity of its balance in ¢.
- $\mathcal{X}_i^d(e)$: A challenge by $\mathcal{P}_i$ against the integrity of an off-chain transfer delivery in ¢.

Consequently, $A_i(e)$ is calculated as follows in $\mathcal{B}$:

$$A_i(e) = A_i(e-1) + \mathcal{D}_i(e-1) + R_i(e-1) - \mathcal{W}_i(e-1) - S_i(e-1) \quad (1)$$

Additionally, For every *eon e*, $\mathcal{B}^G(e)$ can store a commitment by $O_{¢}$ to the contents of $\mathcal{B}^L(e-1)$. The underlying data-structure of this commitment is explained in Section 3.4.1.

3.4 $\mathcal{T}_{¢}$ Periodic Commitments

In this section we describe the data-structures and message formats that enable the efficient provable integrity of a NOCUST *eon*.

3.4.1 Synchronization Tree-structure. To provably account for the allotted balances $A_i(e)$ of each $\mathcal{P}_i$ at the beginning of an *eon e*, we design a novel *Merklelized interval tree* $\mathcal{T}_{¢}(e)$. The Merkleized interval tree is similar to the augmented merkle tree proposed by Luu *et al.* [32], yet built and utilized in completely different means. The nodes in this Merkle tree [33] are augmented to store the user balances in an efficient manner that allows $\mathcal{V}_{¢}$ to securely verify the correct allotment of funds by $O_{¢}$. A node $t_n(e)$ of $\mathcal{T}_{¢}$ is structured as defined in Equation 2.

$$t_n(e) =< \texttt{offset}_n(e), \texttt{information}_n(e), \texttt{allotment}_n(e) > \quad (2)$$

`offset` and `allotment` are both numeric values, while `information` is a cryptographic commitment to the information contained within this node. The values for these fields are defined differently for leaves and internal nodes as follows.

A leaf $t_i(e)$ is used to represent the off-chain account of a $\mathcal{P}_i$ at *eon e*, whereby $\texttt{allotment}_i(e)$ is equal to $A_i(e)$ (cf. Section 3.3.3), and $\texttt{offset}_i(e)$ corresponds to the sum of the allotted balances of all participants ordered before $\mathcal{P}_i$ (cf. Equation 4).

$$\texttt{allotment}_i(e) = A_i(e) \quad (3)$$

$$\texttt{offset}_i(e) = \sum_{j<i} \texttt{allotment}_j(e) \quad (4)$$

$\texttt{information}_i(e)$ is composed of the cryptographic hash of the blockchain address of $\mathcal{P}_i$ and the commitment of the last balance update agreed to by $\mathcal{P}_i$ in the previous *eon*. More precisely,

$$\texttt{information}_i(e) = \{\texttt{address}_i, \texttt{update}_i(e-1)\} \quad (5)$$

where $\texttt{update}_i(e)$ represents the last state update of the off-chain account of $\mathcal{P}_i$ at *eon e* as described in Section 3.4.3.

An internal node $t_u(e)$, with a left child $t_p(e)$ and a right child $t_q(e)$, is constructed per (cf. Equation 6) and (cf. Equation 7) :

$$\texttt{allotment}_u(e) = \texttt{allotment}_p(e) + \texttt{allotment}_q(e) \quad (6)$$

$$\texttt{offset}_u(e) = \texttt{offset}_p(e) \quad (7)$$

$\texttt{information}_u(e)$ is a cryptographic commitment similar to that of an internal node of a Merkle Tree but with the addition of $\texttt{offset}_q(e)$ as a third middle value.

$$\texttt{information}_u(e) = \{t_p(e), \texttt{offset}_q(e), t_q(e)\} \quad (8)$$

It's important to note that the middle value of $\texttt{offset}_q(e)$ is interchangeable with that of $\texttt{offset}_p(e) + \texttt{allotment}_p(e)$ as they should be equal in correct instances of this structure.

3.4.2 Proof of exclusive allotment. For each $\mathcal{P}_i$ included in $\mathcal{T}_{¢}(e)$, a proof of exclusive allotment $\tau_i(e)$ can be constructed. The main goal of this construct is to prove that $\mathcal{P}_i$ *exclusively* owns an allotment of size $A_i(e)$ within the allotment covered by $\mathcal{T}_{¢}(e)$.

$\tau_i(e)$ is constructed similar to a regular merkle tree membership proof, whereby the nodes adjacent to the path from the root to the leaf constitute the membership proof hash chain. However, in addition to the hashes of the nodes in the membership proof, a boundary value Ω is required for each node:

$$\Omega(t_i(e), t_n(e)) = \begin{cases} \texttt{offset}_n(e) & t_n(e) \text{ is a left child} \\ \texttt{offset}_n(e) + \texttt{allotment}_n(e) & t_n(e) \text{ is a right child} \end{cases} \tag{9}$$

The procedure of verifying $\tau_i(e)$ is similar to that of verifying set membership in a merkle tree but the node reconstruction is done so according to the definitions the $\mathcal{T}_{\not\subset}$ structure in Section 3.4.1 in conjunction with the Ω values. This bounds the size of a $\tau_i(e)$ to $O(\log|\mathcal{P}|)$.

3.4.3 Monotonic $\mathcal{P}$-State Structure. The information in $\texttt{update}_i(e)$ contained in $\mathcal{T}_{\not\subset}(e)$ is structured as follows:

$$\texttt{update}_i(e) = \{T_i(e), S_i(e), R_i(e)\} \tag{10}$$

$T_i(e)$ is committed to using a merkle tree where the leaves are the individual transfers through $O_{\not\subset}$ authorized by $\mathcal{P}_i$ during *eon* e. A $O_{\not\subset}$ transfer $\mathfrak{T}$ is a tuple of the following information:

$$< eon, sender, recipient, amount > \tag{11}$$

The merkle tree used to create the commitment for $T_i(e)$ need not be augmented. We refer to the standard merkle tree proof of membership that $\mathfrak{T} \in T_i(e)$ as $\lambda(\mathfrak{T} \in T_i(e))$.

3.5 $\mathcal{V}_{\not\subset}$ On-chain Verifier

The on-chain component $\mathcal{V}_{\not\subset}$ acts as the bridge between the $O_{\not\subset}$ ledger $\mathcal{B}^L$ and the $\mathcal{BC}$ ledger $\mathcal{B}^G$. Its procedures are assumed to be executed honestly by $\mathcal{BC}$, and it supports the following operations:

3.5.1 ***Commit*** *$\mathcal{T}_{\not\subset}(e)$.* Committing to a $\mathcal{T}_{\not\subset}(e)$ may only be done once per *eon* e during its first *epoch* by $O_{\not\subset}$. The commitment requires only submission of the root node of $\mathcal{T}_{\not\subset}(e)$.

The commitment procedure involves no validation on `information`, but the following requirements exist on the `offset` and `allotment` of the root node:

$$\texttt{offset}_{root}(e) = 0 \tag{12}$$

$$\texttt{allotment}_{root}(e) = \texttt{allotment}_{root}(e-1) + \mathcal{D}(e-1) - \mathcal{W}(e-1) \tag{13}$$

After validation, $\mathcal{V}_{\not\subset}$ stores the node information, making it available to any $\mathcal{P}$ or any other $\mathcal{V}_{\not\subset}$ procedure.

Preconditions:

- $O_{\not\subset}$ must not have committed to $\mathcal{T}_{\not\subset}(e)$
- $\not\subset$ must not have entered recovery

Input: $t_{root}(e)$

(1) Verify conditions of Equation 12 and 13 on $t_{root}(e)$
(2) Store $t_{root}(e)$ as the commitment to $\mathcal{T}_{\not\subset}(e)$

3.5.2 ***Verify*** *$\tau_i(e)$.* This verification procedure enables $\mathcal{V}_{\not\subset}$ to verify a $\tau_i(e)$ for any e in which $O_{\not\subset}$ had committed to a $\mathcal{T}_{\not\subset}(e)$. This validation acts as a foundation for the security of NOCUST.

Preconditions:

- $O_{\not\subset}$ must have committed to $\mathcal{T}_{\not\subset}(e)$

Input: $\tau_i(e)$

(1) Reconstruct $t'_{root}(e)$ from $\tau_i(e)$
(2) output true iff $t'_{root}(e) = t_{root}(e)$

3.5.3 ***Receive Deposit*** *$\mathcal{D}_i(e)$.* For a $\mathcal{P}_i$ to make a deposit into $\not\subset$, it would simply send a transfer in the $\mathcal{BC}$ ledger with $\mathcal{V}_{\not\subset}$ as the recipient. The only requirement on $\mathcal{V}_{\not\subset}$ is then that it adds the value of the transfer to $\mathcal{D}_i(e)$, where e is the current *eon*.

Preconditions:

- $\not\subset$ must not have entered recovery

Input: $\mathcal{BC}$ transfer $\mathtt{T}$ from $\mathcal{P}_i$ to $\mathcal{V}_{\not\subset}$

(1) Set $\mathcal{D}_i(e)$ to $\mathcal{D}_i(e) + \mathtt{T}.\texttt{amount}$

3.5.4 ***Initiate Withdrawal*** *$\mathcal{W}_i(e)$.* A $\mathcal{P}_i$ can initiate a withdrawal from $\not\subset$ by submitting a request to $\mathcal{V}_{\not\subset}$. This request consists of the amount, to be withdrawn once the request is confirmed, and of $\tau_i(e-1)$, where e is the current *eon*. If any previous withdrawal had been successfully issued but not yet confirmed, $\mathcal{V}_{\not\subset}$ must reject this request.

After calling its own validation procedure, with the stored $t_{root}(e-1)$ as reference and $\tau_i(e-1)$ as input, upon a successful result $\mathcal{V}_{\not\subset}$ is required to set $\mathcal{W}_i(e)$ to the requested amount, while upon validation failure, or if $A_i(e-1)$ is less than the requested amount, $\mathcal{V}_{\not\subset}$ should reject the request.

Preconditions:

- $\not\subset$ must not have entered recovery
- $\mathcal{P}_i$ may not have any other pending withdrawals

Input: $\tau_i(e-1)$, amount to be withdrawn $\mathtt{w}$

(1) Validate $\tau_i(e-1)$
(2) Validate $\mathtt{w} \le A_i(e-1)$
(3) Set $\mathcal{W}_i(e)$ to w

3.5.5 ***Cancel Withdrawal*** *$\mathcal{W}_i(e)$.* A malicious $\mathcal{P}_i$ may request to withdraw funds from $\mathcal{V}_{\not\subset}$ after having spent them through $O_{\not\subset}$ during e. The $O_{\not\subset}$ can provide $\tau_i(e-1)$ and an $\texttt{update}_i$ signed by $\mathcal{P}_i$ to $\mathcal{V}_{\not\subset}$ to prove that $\mathcal{P}_i$'s balance had fallen below the requested amount and cancel $\mathcal{W}_i(e)$. This procedure can be augmented with a punishment against $\mathcal{P}_i$ as a disincentive for misbehavior.

Preconditions:

- $\not\subset$ must not have entered recovery

Input: $\mathcal{P}_i$, signed $\texttt{update}_i(e)$ or $\texttt{update}_i(e-1)$, $\tau_i(e-1)$

(1) Verify $Sig_i(\texttt{update}_i)$
(2) Validate $\tau_i(e-1)$
(3) Confirm $\mathcal{W}_i(e) > A_i(e-1) + \texttt{update}_i.R$- $\texttt{update}_i.S$
(4) Set $\mathcal{W}_i(e)$ to 0

3.5.6 ***Confirm Withdrawal*** *$\mathcal{W}_i(e)$.* In *eon* e, a withdrawal request can be confirmed if it had not been provably cancelled by $O_{\not\subset}$ and if it was scheduled in *eon* $e-2$ and the first *epoch* has passed, or if it had been scheduled in an *eon* $\le e-3$.

Upon confirmation of a withdrawal, $\mathcal{V}_{\not\subset}$ issues a transfer from the balance pool it manages in favor of $\mathcal{P}_i$ with the requested amount.

Preconditions:

- $W_i(s) > 0$ for some $s \le e-2$

Input: none

(1) Reject if $s = e-2$ and the first *epoch* of e has not passed
(2) Transfer $W_i(s)$ to $\mathcal{P}_i$ on $\mathcal{BC}$
(3) Set $W_i(s)$ to 0

3.5.7 ***Open Balance Update Challenge*** $\mathcal{X}_i^b(e)$.
Given a $\tau_i(e-1)$ and an `update`$_i(e-1)$ signed by $\mathcal{O}_{\not\subset}$ as inputs from a $\mathcal{P}_i$, the $\mathcal{V}_{\not\subset}$ challenge procedure requires that the hub provides a satisfying $\tau_i(e)$ $\mathcal{V}_{\not\subset}$ before an *epoch* passes. Otherwise, $\not\subset$ is shut down, and all transactions since the beginning of $e-1$ are reverted.

Preconditions:

- $\not\subset$ must not have entered recovery

Input: At least one of $\tau_i(e-1)$ and `update`$_i(e-1)$

- Verify $\tau_i(e-1)$, or $A_i(e-1) = 0$
- Verify $Sig_O($`update`$_i(e-1))$, or $R_i(e-1) = S_i(e-1) = 0$
- Store expected $A_i(e)$ in $\mathcal{X}_i^b(e)$

3.5.8 ***Close Balance Update Challenge*** $\mathcal{X}_i^b(e)$.
Given a valid $\tau_i(e)$ as input from $\mathcal{O}_{\not\subset}$, $\mathcal{V}_{\not\subset}$ marks $\mathcal{X}_i^b(e)$ as closed if it were open within the last *epoch*.

Preconditions:

- $\not\subset$ must not have entered recovery
- $\exists\, \mathcal{X}_i^b(e)$ not older than an *epoch*

Input: $\tau_i(e)$, `update`$_i(e-1)$

(1) Verify $\tau_i(e)$
(2) Verify $Sig_i($`update`$_i(e-1))$
(3) Verify $Sig_O($`update`$_i(e-1))$
(4) Verify `update`$_i(e-1)$ is at least as recent as in $\mathcal{X}_i^b(e)$
(5) Validate that $\tau_i(e)$ ratifies `update`$_i(e-1)$
(6) Mark $\mathcal{X}_i^b(e)$ closed

3.5.9 ***Open Transfer Delivery Challenge*** $\mathcal{X}_i^d(e)$.
Given an `update`$_j(e-1)$ signed by $\mathcal{O}_{\not\subset}$, and a transfer $\mathfrak{T}_i^j(e-1) \in T_j(e-1)$ as inputs from $\mathcal{P}_i$ or $\mathcal{P}_j$, the $\mathcal{V}_{\not\subset}$ delivery challenge procedure requires that the hub provide a satisfying $\tau_i(e)$ and $\lambda(\mathfrak{T}_i^j(e-1) \in T_i(e-1))$ to $\mathcal{V}_{\not\subset}$ before an *epoch* passes. Otherwise, $\not\subset$ is shut down, and all transactions since the beginning of $e-1$ are reverted.

Preconditions:

- $\not\subset$ must not have entered recovery

Input: `update`$_j(e-1)$, $\mathfrak{T}_j^i(e-1)$, $\lambda(\mathfrak{T}_i^j(e-1) \in T_j(e-1))$

- Verify $Sig_O($`update`$_j(e-1))$
- Verify $\lambda(\mathfrak{T}_i^j(e-1) \in T_j(e-1))$
- Store $\mathfrak{T}_i^j(e-1)$ in $\mathcal{X}_i^d(e)$

3.5.10 ***Close Transfer Delivery Challenge*** $\mathcal{X}_i^d(e)$.
Given a valid $\tau_i(e)$, `update`$_i(e-1)$ and $\lambda(\mathfrak{T}_i^j(e-1) \in T_i(e-1))$ as input from $\mathcal{O}_{\not\subset}$, $\mathcal{V}_{\not\subset}$ marks $\mathcal{X}_i^d(e)$ as closed if it were open within the last *epoch*.

Preconditions:

- $\not\subset$ must not have entered recovery
- $\exists\, \mathcal{X}_i^d(e)$ not older than an *epoch*

Input: $\tau_i(e)$, `update`$_i(e-1)$, $\lambda(\mathfrak{T}_i^j(e-1) \in T_i(e-1))$

(1) Verify $\tau_i(e)$
(2) Validate $Sig_i($`update`$_i(e-1))$
(3) Validate that $\tau_i(e)$ ratifies `update`$_i(e-1)$
(4) Validate $\lambda(\mathfrak{T}_i^j(e-1) \in T_i(e-1))$
(5) Mark $\mathcal{X}_i^d(e)$ closed

3.5.11 ***Recover Funds***.
Had any $\mathcal{X}_i^b(e-1)$ or $\mathcal{X}_i^d(e-1)$ not been closed for any i within one *epoch*, or if $\mathcal{O}_{\not\subset}$ fails to commit to $\mathcal{T}_{\not\subset}(e)$ within the first *epoch* of e, $\not\subset$ is considered to have shut down and gone into recovery mode, whereby any $\mathcal{P}_i$ may withdraw all their off-chain funds as of the end of $e-2$, and all their on-chain deposits starting from $e-1$ by providing $\tau_i(e-2)$ to $\mathcal{V}_{\not\subset}$.

Preconditions:

- $\not\subset$ must have entered recovery
- $\mathcal{P}_i$ may not have previously recovered its funds

Input: $\tau_i(e-2)$

(1) Validate $\tau_i(e-2)$
(2) Transfer $A_i(e-2) + \mathcal{D}_i(e-2) + \mathcal{D}_i(e-1)$ to $\mathcal{P}_i$
(3) Mark $\mathcal{P}_i$ as recovered

3.6 $\mathcal{O}_{\not\subset}$ Off-chain Operator

The off-chain component $\mathcal{O}_{\not\subset}$ acts as the facilitator of transfers between members of $\mathcal{P}$, and is designed to behave as follows:

3.6.1 ***Admit*** $\mathcal{P}_i$.
On request to enter the managed $\not\subset$ instance from a participant, $\mathcal{O}_{\not\subset}$ need only append the participant to $\mathcal{P}$ and acknowledgement its `update`$_i(e)$ reflecting an empty balance by providing a countersignature on it.

3.6.2 ***Create*** $\mathcal{T}_{\not\subset}(e)$.
After an *eon* $e-1$ is over, $\mathcal{O}_{\not\subset}$ creates $\mathcal{T}_{\not\subset}(e)$ by using all confirmed transfer information in $e-1$. This means that for each $\mathcal{P}_i$, the last `update`$_i(e-1)$ ratified by $\mathcal{O}_{\not\subset}$ would be used to construct $\mathcal{T}_{\not\subset}(e)$ as described in Section 3.4.1.

3.6.3 ***Commit*** $\mathcal{T}_{\not\subset}(e)$.
After the creation of $\mathcal{T}_{\not\subset}(e)$, $\mathcal{O}_{\not\subset}$ needs to commit $t_{root}(e)$ to $\mathcal{V}_{\not\subset}$ within the first *epoch* of e, or be halted in $\mathcal{V}_{\not\subset}$.

3.6.4 ***Provide*** $\tau_i(e)$.
After constructing $\mathcal{T}_{\not\subset}(e)$, $\mathcal{O}_{\not\subset}$ communicates each $\tau_i(e)$ to its respective $\mathcal{P}_i$ such that $\mathcal{P}_i$ can verify the integrity of its off-chain balance or issue a challenge if need be.

3.6.5 ***Deliver Transfers***.
$\mathcal{O}_{\not\subset}$ requires a transfer $\mathfrak{T}_j^i(e)$ from a $\mathcal{P}_i$ to a $\mathcal{P}_j$ to proceed as follows:

(1) $\mathcal{P}_i$ sends a new signed `update`$_i(e)$ to $\mathcal{O}_{\not\subset}$ with $T_i(e) \cup \mathfrak{T}_j^i(e)$.
(2) $\mathcal{P}_j$ sends a new signed `update`$_j(e)$ to $\mathcal{O}_{\not\subset}$ with $T_j(e) \cup \mathfrak{T}_j^i(e)$.
(3) $\mathcal{O}_{\not\subset}$ ratifies both `update`$_i(e)$ and `update`$_j(e)$ and sends its signatures to $\mathcal{P}_i$ and $\mathcal{P}_j$ respectively.

$\mathcal{O}_{\not\subset}$ must enforce that a $\mathcal{P}_i$ may only have one transfer ongoing at a time. Abortion prior to the last confirmation by $\mathcal{O}_{\not\subset}$ may be signaled via peripheral messages.

3.6.6 ***Credit Deposits*** $\mathcal{D}_i(e)$.
$\mathcal{O}_{\not\subset}$ is required to monitor $\mathcal{V}_{\not\subset}$ and properly credit all deposits $\mathcal{D}_i(e)$ made by every $\mathcal{P}_i$ or face balance update challenges in the next *eon*. This is done by simply increasing the allotment $A_i(e+1)$ for a deposit made in e such that Equation 13 holds for $e+1$.

3.6.7 ***Moderate Withdrawals*** $\mathcal{W}_i(e)$.
A malicious $\mathcal{P}_i$ may interact directly with $\mathcal{V}_{\not\subset}$ to initiate a withdrawal of funds that were confirmed in $e-1$ but were spent in e off-chain. In such cases, $\mathcal{O}_{\not\subset}$ must use the withdrawal cancellation procedure in $\mathcal{V}_{\not\subset}$ before the end of e, or risk the inability to construct an acceptable commitment in $e+1$. When the withdrawal is correct, however, $\mathcal{O}_{\not\subset}$ must debit the allotment $A_i(e+1)$ such that Equation 13 holds for $e+1$.

3.6.8 ***Close challenges*** $\mathcal{X}_i^b(e)$, $\mathcal{X}_i^d(e)$. A $\mathcal{P}_i$ may issue a challenge via $\mathcal{V}_{\not{C}}$ at any moment. $O_{\not{C}}$ needs to monitor $\mathcal{V}_{\not{C}}$ for these challenges and issue appropriate responses to close them, or risk being halted. It is guaranteed that an honest $O_{\not{C}}$ will always have the information required to construct a valid call to $\mathcal{V}_{\not{C}}$ to close invalid challenges.

3.7 $\mathcal{P}$ Clients

Members of $\mathcal{P}$ are the main parties interested in transferring funds to each other in $\not{C}$, and are designed to behave as follows:

3.7.1 ***Join*** $\not{C}$. A $\mathcal{P}_i$ wishing to join a $\not{C}$ instance during *eon* e need only do so through $O_{\not{C}}$ by providing a signed $\texttt{update}_i(e)$ and waiting for acknowledgement in the form of a countersignature. The update should reflect an empty account within the $\not{C}$ instance.

3.7.2 ***Audit*** $\tau_i(e)$. $\mathcal{P}_i$ must ensure that it always receives a valid $\tau_i(e)$ (acceptable by $\mathcal{V}_{\not{C}}$) every *eon* e from $O_{\not{C}}$ to maintain custody of its funds throughout the time progression and enforce correct transfer delivery by $O_{\not{C}}$.

3.7.3 ***Send Transfer.*** A $\mathcal{P}_i$ wishing to enact a $\mathfrak{T}_j^i(e)$ to a $\mathcal{P}_j$ during eon e sends a signed $\texttt{update}_i(e)$ to $O_{\not{C}}$ and notifies $\mathcal{P}_j$ that they send a signed $\texttt{update}_j(e)$ to $O_{\not{C}}$ that reflects receipt. $\mathcal{P}_i$ should expect $O_{\not{C}}$ to return its own signature on $\texttt{update}_i(e)$, after $\mathcal{P}_j$ submits its receipt to $O_{\not{C}}$, before proceeding with sending or receiving further transfers. Moreover, $\mathcal{P}_i$ may not attempt to initiate any other transfers until $O_{\not{C}}$ countersigns $\texttt{update}_i(e)$.

3.7.4 ***Receive Transfer.*** A $\mathcal{P}_i$ notified of a transfer $\mathfrak{T}_i^j(e)$ by a $\mathcal{P}_j$ should hand over a signed $\texttt{update}_i(e)$ to $O_{\not{C}}$ reflecting receipt and wait for a countersignature on $\texttt{update}_i(e)$ by $O_{\not{C}}$ to confirm delivery commitment before proceeding with further transfers. Again, $\mathcal{P}_i$ may not initiate any other transfers until $O_{\not{C}}$ countersigns $\texttt{update}_i(e)$.

3.7.5 ***Deposit*** $\mathcal{D}_i(e)$. Clients that wish to deposit into $\not{C}$ must do so only while in possession of a $\tau_i(e)$, or a ratified $\texttt{update}_i(e)$ if this is the first *eon* for $\mathcal{P}_i$ in $\not{C}$, and only if $\not{C}$ is not in recovery. The deposit is done through sending a $\mathcal{BC}$ transaction to $\mathcal{V}_{\not{C}}$.

3.7.6 ***Withdrawal*** $\mathcal{W}_i(e)$. To withdraw funds during *eon* e, clients utilize their $\tau_i(e-1)$ and not attempt to overdraw beyond their minimum within the current and past *eon*, or face their withdrawals being cancelled by an honest $O_{\not{C}}$, or cancelled by the halt of $\not{C}$. After the first *epoch* of $e+2$ passes successfully, clients may claim $\mathcal{W}_i(e)$ on $\mathcal{BC}$ using $\mathcal{V}_{\not{C}}$.

3.7.7 ***Issue*** $\mathcal{X}_i^b(e)$. If $O_{\not{C}}$ does not provide a valid $\tau_i(e)$ after commitment to $\mathcal{T}_{\not{C}}(e)$, a $\mathcal{P}_i$ should issue a $\mathcal{X}_i^b(e)$ using $\mathcal{V}_{\not{C}}$.

3.7.8 ***Issue*** $\mathcal{X}_i^d(e)$. When shown proof of debit (signed $\texttt{update}_j(e)$ by $O_{\not{C}}$) not reflected by an authorized credit in $\tau_i(e)$, a $\mathcal{P}_i$ should issue a $\mathcal{X}_i^d(e)$ to $\mathcal{V}_{\not{C}}$.

Unless $\mathcal{P}_j$ is malicious, $O_{\not{C}}$ will not be able to close $\mathcal{X}_i^d(e)$. $\mathcal{P}_j$ should also issue the challenge in case of $\mathcal{P}_i$'s noncooperation or if a receipt confirmation is not provided.

3.7.9 ***Recover Funds.*** Upon $O_{\not{C}}$'s failure to close any challenge within one *epoch* and before e ends, $\not{C}$'s time progression stops at e and it enters into recovery. Every $\mathcal{P}_i$ will need to recover its confirmed off-chain funds through $\mathcal{V}_{\not{C}}$.

4 SECURITY ANALYSIS

In this section we will analyze the security guarantees of NOCUST, assuming that the underlying layer $\mathcal{BC}$ may serve as a recourse for settling disputes on the integrity of $\not{C}$. In this model we also assume that there are costs associated with $\mathcal{BC}$ settlement, such as a gas fee paid to perform smart contract operations or transactions in the Ethereum network. We consider these settlement expenses as external to the balance of a $\mathcal{P}_i$ in the system, but have also designed NOCUST to minimize these expenses such that the amount of information required for dispute settlement is feasibly transmittable, as bounded in Section 3.4. The NOCUST protocol is designed to prevent any honest member of $\mathcal{P}$ from losing any funds despite a strong set of adversarial capabilities.

4.1 Threat Model

We assume that there are two classes of users in NOCUST: (i) $O_{\not{C}}$ operators and (ii) $\mathcal{P}$ participants which can both receive incoming and send outgoing transactions. We will assume the existence of an irrational adversary willing to sustain financial losses in order to cause honest parties to lose some or all of their funds in $\not{C}$. This irrational adversary may seize control of $O_{\not{C}}$, some or all but one of $\mathcal{P}$, or a combination thereof, in order to attack an honest $\mathcal{P}_i$ not under its control. The adversary has full control of the identities associated with the compromised parties and may authorize any messages on their behalf or front-run any user input, but cannot violate the integrity of the honest users' identities. Moreover, an adversary may launch denial of service attacks that degrade the off-chain communication between $O_{\not{C}}$ and members of $\mathcal{P}$, but may not compromise an honest $\mathcal{P}_i$'s communication with $\mathcal{BC}$, respectively $\mathcal{V}_{\not{C}}$. In the following discussion, we define malicious behavior as that which aims to cause an honest $\mathcal{P}_i$ to lose control of some or all of its funds in $\not{C}$ or cause an honest $O_{\not{C}}$ to be forcibly shut down by $\mathcal{V}_{\not{C}}$.

4.2 Guarantees

In this section we explain how under the stated threat model, an honest $\mathcal{P}_i$ can securely maintain custody of its funds and ensure that its enacted transfers are correctly delivered within $\not{C}$, but will not be able to forcibly enact any new transfer $\mathfrak{T}_j^i(e)$ in the system without facilitation by $O_{\not{C}}$. We also demonstrate how an honest $O_{\not{C}}$ can sustain service under the malice of a subset of $\mathcal{P}$. We prove the security guarantees of NOCUST through proving that an honest $\mathcal{P}_i$ or honest $O_{\not{C}}$ following the prescribed protocol may not end in a state where they cannot utilize $\mathcal{V}_{\not{C}}$ to enforce the integrity of $\not{C}$. We refrain from building a single comprehensive model of the system due to the many interactions present, and instead break down the system into different components and prove their security properties. We argue that under the stated system model in Section 3.1 it suffices to prove the sanity of agent behavior in NOCUST due to the presence of $\mathcal{V}_{\not{C}}$ and the feasibility of deploying its functionality to operate honestly on $\mathcal{BC}$.

4.2.1 *Exclusive Allotment.*

An important element in NOCUST is the dependence on a valid τ_i to guarantee to a $\mathcal{P}_i$ the exclusive allotment of a portion of size A_i in the funds managed by $\mathcal{V}_{¢}$ to $\mathcal{P}_i$ alone, therefore, we proceed to prove that no valid instance of $\mathcal{T}_{¢}$ may contain two overlapping allotments, and therefore that $\mathcal{V}_{¢}$ cannot accept an invalid τ_i.

Proof. We proceed to prove that no valid instance of $\mathcal{T}_{¢}$ may be used to construct a τ_i that permits a non-exclusive allotment by contradiction. Assume a valid instance of $\mathcal{T}_{¢}$, and without loss of generality let t_x and t_y be two successive nodes ($y > x$) within $\mathcal{T}_{¢}$ that have overlapping allotments, where $\texttt{offset}_y < \texttt{offset}_x + \texttt{allotment}_x$.

Let t_u be their least common ancestor with t_p and t_q as its direct children such that t_p is an ancestor of t_x, and t_q of t_y. Without loss of generality, assume t_p and t_q are correctly reconstructible from τ_x and τ_y respectively.

Given τ_x, constructing t_u on the path up to t_{root} will be performed with knowledge of $\texttt{offset}_p$ and $\texttt{allotment}_p$ (from reconstructing t_p) and the boundary value and commitment of t_q supplied in τ_x.

$$\Omega(t_u, t_q) = \texttt{offset}_q + \texttt{allotment}_q \tag{14}$$

Recall the definition in Section 3.4.1. As $\texttt{offset}_q$ is interchangeable with $\texttt{offset}_p + \texttt{allotment}_p$, reconstructing t_u will need to be performed as follows due to the lack of presence of $\texttt{offset}_q$ in τ_x by substitution in equation 8 as follows:

$$\texttt{information}_u = \{t_p, \texttt{offset}_p + \texttt{allotment}_p, t_q\} \tag{15}$$

Given the correctness of the sub-tree of t_p in isolation, it follows that $\texttt{offset}_p + \texttt{allotment}_p = \texttt{offset}_x + \texttt{allotment}_x$, and therefore, assuming $\texttt{offset}_q$ was used in the original commitment to the considered instance of $\mathcal{T}_{¢}$, the reconstructed t_u will not match, and the remaining trail of reconstructed nodes in τ_x[1] will lead to a $t'_{root} \neq t_{root}$, violating the assumption that the instance under consideration is a valid $\mathcal{T}_{¢}$ and that τ_x is acceptable □

4.2.2 *Balance Custody.*

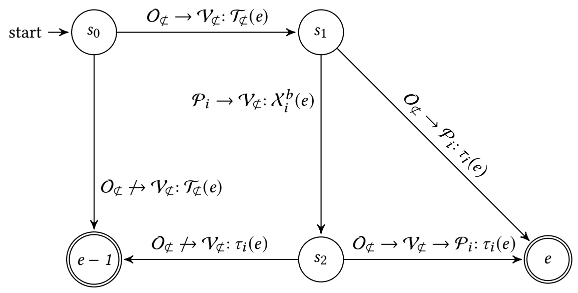
An honest participant $\mathcal{P}_i$ of ¢ maintains custody of its balance in the system because it may always resort to $\mathcal{V}_{¢}$ in case $O_{¢}$ does not provide a valid $\tau_i(e)$ for any *eon e*.

A participant must maintain knowledge about its state to preserve its ability to utilize $\mathcal{V}_{¢}$ for dispute regardless of how the other participants and $O_{¢}$ behave. By keeping track of every authorized $\texttt{update}_i(e)$ and every $\tau_i(e)$ from $O_{¢}$ for each *eon e* that passes after $\mathcal{P}_i$ entered ¢, $\mathcal{P}_i$ may always open $\mathcal{X}_i^b(e)$ in case $O_{¢}$ fails to provide a $\tau_i(e)$ with a correct exclusive $A_i(e)$. This guarantees that $\mathcal{P}_i$ is always able to enforce a provably correct update by $O_{¢}$ to its state in $\mathcal{T}_{¢}(e)$, or halt ¢.

As a withdrawal $\mathcal{W}_i(e)$ may only be initiated through $\mathcal{V}_{¢}$, which requires a verifiable $\tau_i(e-1)$ to be submitted by $\mathcal{P}_i$, no other $\mathcal{P}_j$ may attempt to initiate a claim for any portion of $A_i(e-1)$. Moreover, a $\tau_i(e)$ may not be utilized to initiate any withdrawals until *eon* $e+1$ commences with no open challenges against the integrity of $\mathcal{T}_{¢}(e)$. This guarantees that only uncontested exclusive balance allotments in a correct $\mathcal{T}_{¢}$ instance may be used to enact withdrawals that may not be interrupted by $O_{¢}$ unless they attempt a double spend.

[1]A symmetric argument can be made for τ_y

Proof. We proceed to prove how an honest $\mathcal{P}_i$ in NOCUST can protect its funds through modelling the state of a $\mathcal{P}_i$'s custody as a finite state machine whereby $\mathcal{P}_i$ may always reach a custodian state. A $\mathcal{P}_i$ is considered a non-custodian in *eon e* if *e − 1* had passed successfully (¢ did not enter recovery) without $\mathcal{P}_i$ learning a valid $\tau_i(e-1)$ that exclusively accounts for its confirmed balance, assuming $\mathcal{P}_i$ joined ¢ prior to *e − 1*.



start
s_0
s_1
s_2
$e-1$
e
$O_{¢} \rightarrow \mathcal{V}_{¢}\colon \mathcal{T}_{¢}(e)$
$\mathcal{P}_i \rightarrow \mathcal{V}_{¢}\colon \mathcal{X}_i^b(e)$
$O_{¢} \rightarrow \mathcal{P}_i\colon \tau_i(e)$
$O_{¢} \not\rightarrow \mathcal{V}_{¢}\colon \mathcal{T}_{¢}(e)$
$O_{¢} \not\rightarrow \mathcal{V}_{¢}\colon \tau_i(e)$
$O_{¢} \rightarrow \mathcal{V}_{¢} \rightarrow \mathcal{P}_i\colon \tau_i(e)$

Figure 2: A finite state automaton capturing the custodian state of an honest $\mathcal{P}_i$ during an *eon e*. Given a $O_{¢}$'s commitment to $\mathcal{T}_{¢}$, an honest $\mathcal{P}_i$ may always guarantee its knowledge of a valid $\tau_i(e)$ either cooperatively with $O_{¢}$ or through $\mathcal{V}_{¢}$. Terminal states denote which eon's balance $\mathcal{P}_i$ is given custody of.

It's straightforward to infer from the automaton in Figure 2 that a $\mathcal{P}_i$ may always reach a state of custody from s_1 given that $O_{¢}$ commits to $\mathcal{T}_{¢}(e)$ within the first epoch. Recall that if $O_{¢}$ does not commit to a $\mathcal{T}_{¢}(e)$ within the first *epoch*, the ¢ instance is halted, and therefore $\mathcal{P}_i$ retains custody of the previous allotment $A_i(e-1)$ which it may claim through $\mathcal{V}_{¢}$'s recovery. This may also happen if $O_{¢}$ ignores $\mathcal{P}_i$'s challenge. □

For simplicity we omit states and transitions whereby $\mathcal{P}_i$ does not receive a valid $\tau_i(e)$ from $O_{¢}$ and chooses to not resort to $\mathcal{V}_{¢}$ to demand its broadcast within the next *epoch*, as this behavior does not describe an honest $\mathcal{P}_i$.

It is important to recall what $\mathcal{V}_{¢}$ accepts as a valid response from $O_{¢}$ to a $\mathcal{X}_i^b(e)$ as stated in Section 3.5. The $\texttt{update}_i(e-1)$ in the commitment must be as recent as that submitted in $\mathcal{X}_i^b(e)$ by $\mathcal{P}_i$, and must bear $\mathcal{P}_i$'s signature. This prevents $O_{¢}$ from attempting to commit an outdated state, and provides $\mathcal{P}_i$ sufficient knowledge to enact any future delivery challenges.

4.2.3 *Double-spend Futility.*

In the following discussion we refer to a double spend as any endeavor by the adversary controlling a $\mathcal{P}_i$, with or without control of $O_{¢}$, to attempt any of the following:

(1) Double spend $\mathcal{P}_i$'s balance in $\mathcal{B}^L$
(2) Spend $\mathcal{P}_i$'s balance in $\mathcal{B}^L$ and withdraw it from $\mathcal{B}^G$

In case the adversary lacks control of $O_{¢}$, attempting to double spend only in $\mathcal{B}^L$ will be trivially prevented by an honest $O_{¢}$, and attempted withdrawals in *eon e* from $\mathcal{B}^G$ of funds spent in $e-1$ will also be cancelled by an honest $O_{¢}$ through $\mathcal{V}_{¢}$.

Moreover, even with control of $O_{¢}$, an adversary may not double spend in the current *eon e* and be able to construct a valid $\mathcal{T}_{¢}$ in the next *eon* $e+1$ correctly satisfying every member of $\mathcal{P}$, as a valid instance of $\mathcal{T}_{¢}$ guarantees exclusive allotments, the sizes of which must correspond to the confirmed balances expected by each honest member of $\mathcal{P}$.

Proof. Let $\mathcal{P}_i$ and $O_{\not\subset}$ be under the control of the adversary $\mathcal{A}$ such that the running balance of $\mathcal{P}_i$ during *eon* e is double spent towards a subset of $\mathcal{P}$ whereby Equation 16 holds by the end of e. $\mathcal{A}$ must construct a valid $\mathcal{T}_{\not\subset}$ for $e+1$ to commit the transfers in e and successfully double spend, while avoiding the halt of $\not\subset$ by an honest $\mathcal{P}_j$.

$$A_i(e+1) = A_i(e) + R_i(e) - S_i(e) + \mathcal{D}_i(e) - \mathcal{W}_i(e) < 0 \tag{16}$$

However, using Equation 13, validated by $\mathcal{V}_{\not\subset}$:

$$\begin{aligned}\sum_j A_j(e+1) &= \sum_j A_j(e) + R_j(e) + \mathcal{D}_j(e) - \mathcal{W}_j(e) - S_j(e) \\ &= \sum_j A_j(e) + \sum_j \mathcal{D}_j(e) - \mathcal{W}_j(e) + \sum_j R_j(e) - S_j(e) \\ &= \texttt{allotment}_{root}(e+1) + \sum_j R_j(e) - S_j(e)\end{aligned} \tag{17}$$

With Equation 17 in mind, if $\mathcal{A}$ were double spending in $\mathcal{B}^L$ by not updating $S_i(e)$, then $\sum_j R_j(e) - S_j(e) > 0$ would follow, and $\texttt{allotment}_{root}(e+1) < \sum_j A_j(e+1)$ would lead to a challenge in $e+1$ by the affected honest $\mathcal{P}_j$ whose allotment is incorrect, foiling as well any concurrent double spend in $\mathcal{B}^G$.

Moreover, if $\mathcal{A}$ were double spending in $\mathcal{B}^L$ and updating $S_i(e)$ such that $\sum_j R_j(e) - S_j(e) = 0$, and/or double spending through $\mathcal{W}_i(e)$ in $\mathcal{B}^G$, then an $\texttt{allotment}_{root}(e+1)$ would be rejected by $\mathcal{V}_{\not\subset}$ in violation of Equation 13. □

4.2.4 Operational Integrity. An adversary in control of some participants in $\mathcal{P}$ may maliciously open a set of challenges against $O_{\not\subset}$ using $\mathcal{V}_{\not\subset}$. In NOCUST, there is no way for $\mathcal{V}_{\not\subset}$ to verify whether a participant is opening a challenge maliciously or not, and therefore $O_{\not\subset}$ must answer every challenge opened. However, it is guaranteed that an honest $O_{\not\subset}$ is able to close any challenge opened in $\mathcal{V}_{\not\subset}$, and a dishonest $O_{\not\subset}$ that misconstructs a $\mathcal{T}_{\not\subset}(e)$ will not be able to answer correct challenges in e.

The information required by an honest $O_{\not\subset}$ to close a $\mathcal{X}_i^b(e)$ is $\tau_i(e)$, which is constructed by $O_{\not\subset}$, and the $\texttt{update}_i(e-1)$ used in the construction. The additional piece of information required to close a $\mathcal{X}_i^d(e)$ is $\lambda(\mathfrak{T}_i^j(e-1) \in T_i(e-1))$, which also has to be known by $O_{\not\subset}$ to construct $\mathcal{T}_{\not\subset}(e)$. Therefore, $O_{\not\subset}$ will always possess sufficient knowledge to close any open challenges, and will successfully do so if the committed $\mathcal{T}_{\not\subset}(e)$ corresponds to the latest ratified contents of $\mathcal{B}^L(e-1)$, given the honesty of $\mathcal{V}_{\not\subset}$ in managing the pool of funds in the $\not\subset$ instance on $\mathcal{BC}$.

Proof. We proceed to prove how honest $O_{\not\subset}$ in NOCUST can maintain functionality under a subset of malicious users in $\mathcal{P}$, and how a dishonest $O_{\not\subset}$ that attempts to compromise transfers will lead to the $\not\subset$ instance being stopped through a proof by case analysis, where we model the provability of a $O_{\not\subset}$'s integrity as a finite state machine whereby transfers are facilitated by $O_{\not\subset}$ in e and committed during $e+1$ in $\mathcal{T}_{\not\subset}(e+1)$. A server is defined as maintaining provable integrity during *eon* e so long as it is able to close any challenge $\mathcal{X}_i(e)$ using $\mathcal{V}_{\not\subset}$.

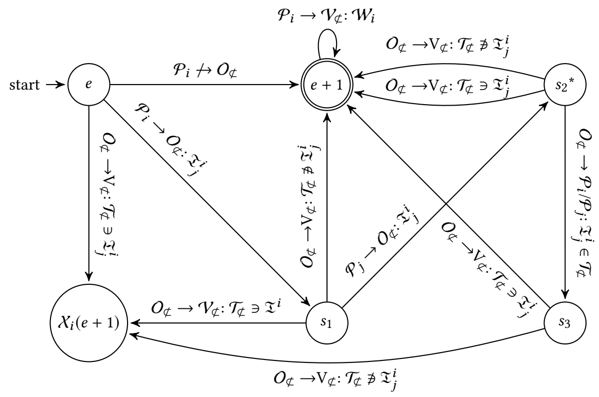

$\mathcal{P}_i \to \mathcal{V}_{\not\subset}: \mathcal{W}_i$
$O_{\not\subset} \to \mathrm{V}_{\not\subset}: \mathcal{T}_{\not\subset} \not\ni \mathfrak{T}_j^i$
start
e
$\mathcal{P}_i \not\to O_{\not\subset}$
$e+1$
$O_{\not\subset} \to \mathrm{V}_{\not\subset}: \mathcal{T}_{\not\subset} \ni \mathfrak{T}_j^i$
$s_2{}^*$
$O_{\not\subset} \to \mathrm{V}_{\not\subset}: \mathcal{T}_{\not\subset} \ni \mathfrak{T}_j^i$
$\mathcal{P}_i \to O_{\not\subset}: \mathfrak{T}_j^i$
$O_{\not\subset} \to \mathrm{V}_{\not\subset}: \mathcal{T}_{\not\subset} \not\ni \mathfrak{T}_j^i$
$\mathcal{P}_j \to O_{\not\subset}: \mathfrak{T}_j^i$
$O_{\not\subset} \to \mathrm{V}_{\not\subset}: \mathcal{T}_{\not\subset} \ni \mathfrak{T}_j^i$
$O_{\not\subset} \to \mathcal{P}_i/\mathcal{P}_j: \mathfrak{T}_j^i \in \mathcal{T}_{\not\subset}$
$\mathcal{X}_i(e+1)$
$O_{\not\subset} \to \mathcal{V}_{\not\subset}: \mathcal{T}_{\not\subset} \ni \mathfrak{T}^i$
s_1
s_3
$O_{\not\subset} \to \mathrm{V}_{\not\subset}: \mathcal{T}_{\not\subset} \not\ni \mathfrak{T}_j^i$

Figure 3: A finite state automaton capturing the provable integrity of $O_{\not\subset}$. An honest $O_{\not\subset}$ may not find itself in a state whereby it cannot prove its integrity in *eon* $e+1$ after committing to its e operations, while a dishonest $O_{\not\subset}$ that attempts to reverse transfers or incorrectly enforce them will find itself unable to do so. *There exists a transition from s_2 to $\mathcal{X}_i(e+1)$ on ($O_{\not\subset} \to \mathcal{V}_{\not\subset}$: $\mathcal{T}_{\not\subset} \ni \mathfrak{T}^i$) omitted for clarity.

The automaton presented in Figure 3 specifies how an honest $O_{\not\subset}$ may always behave in such a way that allows it to retain provable integrity in $e+1$ regardless of the behavior of members of $\mathcal{P}$.

- Given no interactions between $O_{\not\subset}$ and $\mathcal{P}_i$ during e, or given only an $\texttt{update}_i(e)$ signed by $\mathcal{P}_i$, but no $\texttt{update}_j(e)$ signed by $\mathcal{P}_j$, an honest $O_{\not\subset}$ may wait for $\mathcal{P}_j$ or discard $\mathfrak{T}_j^i$. No $\mathcal{X}_i^d(e+1)$ may be opened as $\mathcal{P}_i$ and $\mathcal{P}_j$ would not possess an $\texttt{update}_i(e)$ signed by $O_{\not\subset}$ containing $\mathfrak{T}_j^i(e)$. A $\mathcal{X}_i^b(e+1)$ may be closed with the submission of a $\tau_i(e)$ reflecting the correct $A_i(e+1)$.
- Given an $\texttt{update}_i(e)$ signed by $\mathcal{P}_i$ and an $\texttt{update}_j(e)$ signed by $\mathcal{P}_j$, an honest $O_{\not\subset}$ may discard or synchronize $\mathfrak{T}_j^i(e)$, or commit to its delivery by sending a countersigned $\texttt{update}$ to $\mathcal{P}_i$ and/or $\mathcal{P}_j$ and then must synchronize its delivery in $\mathcal{T}_{\not\subset}(e+1)$. The hub retains sufficient information to close any $\mathcal{X}_i^b(e+1)$ or $\mathcal{X}_i^d(e+1)$ in $\mathcal{V}_{\not\subset}$.
- While in a state of provable integrity, $O_{\not\subset}$ can justifiably cancel any malicious withdrawal by a $\mathcal{P}_i$ using $\mathcal{V}_{\not\subset}$ in order to guarantee being able to satisfy the allotment constraint defined in Equation 13 in $e+1$.

Moreover, a dishonest server which tries to debit a $\mathcal{P}_i$ without authorization, or without crediting the corresponding $\mathcal{P}_j$ in case of a $\mathfrak{T}_j^i$, may not find itself in a state of provable integrity in $e+1$.

- Given no interactions between $O_{\not\subset}$ and $\mathcal{P}_i$ during e, the hub cannot construct a valid $\mathcal{T}_{\not\subset}(e+1)$ containing an $\texttt{update}_i(e)$ signed by $\mathcal{P}_i$. As $O_{\not\subset}$ cannot forge $\mathcal{P}_i$'s signature, it cannot close a $\mathcal{X}_i^b(e+1)$.
- Given only an $\texttt{update}_i(e)$ signed by $\mathcal{P}_i$, the hub cannot construct a valid $\mathcal{T}_{\not\subset}(e+1)$ containing an $\texttt{update}_j(e)$ signed by $\mathcal{P}_j$. A $\mathcal{X}_i^d(e+1)$ on $\mathfrak{T}_j^i(e)$ by a custodian $\mathcal{P}_i$ will not be closeable by $O_{\not\subset}$.

- Once the hub delivers a countersigned $\texttt{update}_i(e)$ and/or $\texttt{update}_j(e)$ ($s_2 \longrightarrow s_3$) to either $\mathcal{P}_i$ or $\mathcal{P}_j$ respectively, it may not back out of enforcing $\mathfrak{T}_j^i(e)$, as $O_{\not{c}}$ will not be able to close a $\mathcal{X}_i^b(e+1)$, and/or $\mathcal{X}_j^b(e+1)$, if it commits an outdated state in $\mathcal{T}_{\not{c}}(e+1)$. □

4.3 Privacy

In this section, we discuss the privacy provisions of NOCUST. In the course of running the protocol, participants acquire proofs of correct operation from and concede state updates to $O_{\not{c}}$, while broadcasting some of that information to $\mathcal{V}_{\not{c}}$ on $\mathcal{BC}$. As the security of the protocol depends on non-repudiation and the forced revelation of information, we explore what each party in the protocol maintains knowledge of and can learn throughout NOCUST.

4.3.1 $O_{\not{c}}$ Knowledge. The operator $O_{\not{c}}$ maintains knowledge of all transfers and balances in $\not{c}$. This is a requirement in NOCUST to enable $O_{\not{c}}$ to facilitate transfers and synchronize between $\mathcal{B}^G$ and $\mathcal{B}^L$ every *eon*, while retaining provable integrity. As such, an adversary in control of $O_{\not{c}}$ has complete knowledge of all off-chain information. Interestingly, at *eon* e, $O_{\not{c}}$ need only maintain knowledge of e and $e-1$ to be able to construct $\mathcal{T}_{\not{c}}(e)$, void any malicious withdrawal $\mathcal{W}_i(e)$ and close any challenge $\mathcal{X}_i^b(e)$, $\mathcal{X}_i^d(e)$ by a $\mathcal{P}_i$. A $O_{\not{c}}$ can erase $e-2$ at the end of *eon* $e-1$ to maintain a form of forward secrecy on the contents of $\mathcal{B}^L$ in $e-2$ without losing operational efficacy, which would retain privacy on all transfers enacted off-chain prior to $e-1$ if $O_{\not{c}}$ were compromised in e.

4.3.2 $\mathcal{P}$ Knowledge. Throughout its participation in a $\not{c}$, a $\mathcal{P}_i$ obtains a $\tau_i(e)$ for every *eon* e, and constructs various $\mathfrak{T}_j^i(e)$ messages for different $\mathcal{P}_j$. A $\tau_i(e)$ reveals the allotment intervals at each height of $\mathcal{T}_{\not{c}}(e)$, but does not reveal individual account addresses or any transfer details. Therefore a $\mathcal{P}_i$ can learn that it has some neighbor account with a certain balance, and learn how the allotted intervals are designated at each level of $\mathcal{T}_{\not{c}}(e)$, without learning the identities of which members of $\mathcal{P}$ these allotments are made to. To enact a transfer $\mathfrak{T}_j^i(e)$, $\mathcal{P}_i$ needs to sign a new $\texttt{update}_i(e)$ and send it to $O_{\not{c}}$, and $\mathcal{P}_j$ needs to sign a new $\texttt{update}_j(e)$ and also send it to $O_{\not{c}}$. $\mathcal{P}_i$ and $\mathcal{P}_j$ need not learn any information about the balance or transfer history of each other to construct these messages, but need to know the full details of the transfer $\mathfrak{T}_j^i(e)$ to ratify it in the state update authorizations they concede. Moreover, to enact a delivery challenge on a transfer $\mathfrak{T}_j^i(e-1)$, $\mathcal{P}_i$ and $\mathcal{P}_j$ need to share $\tau_i(e)$ and $\tau_j(e)$ in order to validate that the transfer was misenforced by $O_{\not{c}}$.

4.3.3 $\mathcal{V}_{\not{c}}$ Knowledge. The privacy of the deposits, challenges and withdrawals conducted through $\mathcal{V}_{\not{c}}$ is scoped by the underlying $\mathcal{BC}$ layer. In NOCUST we do not rely on any privacy of $\mathcal{BC}$ operations. Closing a balance update challenge $\mathcal{X}_i^b(e)$ requires that a $\mathcal{P}_i$ learn the $\tau_i(e)$ used in the closure in order to maintain custody of its account, which reveals $A_i(e)$. Therefore deposits may be used to guess a portion of a $\mathcal{P}_i$'s balance in $\mathcal{B}^L$ without interaction with $O_{\not{c}}$, while closing a $\mathcal{X}_i^b(e)$ reveals $\mathcal{P}_i$'s balance in e, and initiating a $\mathcal{W}_i(e)$ reveals $\mathcal{P}_i$'s balance in $e-1$. It's noteworthy that a $\mathcal{P}_i$ wishing to mask its balance may assume multiple identities on $\mathcal{BC}$ (and consequently in $O_{\not{c}}$) and fragment its deposits and withdrawals over them. However, we leave an extensive analysis for future work.

4.3.4 Comparison to the Privacy Provisions of Two-Party Payment Channels. The guarantees provided in NOCUST are not trivial to compare with those of two-party channel networks. A party's maximum total balance may be inferred from the amount committed to a channel, as the commitment may only be in favor of one party or the other, and the exact amount can be learned from an on-chain withdrawal which necessitates the broadcast of the latest off-chain state. The leakage of the off-chain balance during attempted withdrawals is similar in NOCUST and two-party channels, but inferring the maximum off-chain balance prior to broadcast is not as simple in NOCUST due to the increased number of participants without leakage from $O_{\not{c}}$. As we have not described the enactment of payments across multiple instances of $\not{c}$, we may only draw a rough comparison between linked payments involving more than one intermediary in two-party channel networks and a payment within one $\not{c}$ instance. The added privacy cost of being able to reach different recipients in a two-party channel network is to involve more intermediaries in the payment, while the added privacy cost in the current version of NOCUST is to have one party join the other party's $\not{c}$ instance and still involve only one intermediary. The exact information leakage to be compared depends on the implementation details of the two-party channel and its linked payment mechanism. We leave an extensive comparison for future work.

5 EVALUATION

In the following section, we evaluate the NOCUST model in terms of practicality and usability in the real world and further compare it to the previous state of the art. The basis for the comparison is a payment channel network whereby an entity in place of $\not{c}$ runs and maintains a channel for each member of $\mathcal{P}$.

5.1 Usability

In this section, we discuss the usability of NOCUST as an off-chain payment solution. We explore the advantages in terms of the requirements placed on the operator of a $\not{c}$ instance, and those placed on a $\mathcal{P}_i$.

5.1.1 Collateral Lockup. In a payment channel based network, the locked collateral in each channel is effectively isolated. Therefore, a microtransaction service would have to pre-allocate collateral to each recipient in anticipation of his or her expected transaction volume to eliminate counter-party risk and avoid classification as a credit network. When that collateral is exhausted, the hub would have to retrieve its funds from other channels and consolidate it into recipient channels, either directly through withdrawal or through a highly coordinated rebalancing operation. The first procedure is relatively expensive and requires on-chain operations proportional to the number of users, while the second procedure depends on the channel clients, which are unreliable, and will not guarantee significant consolidation of all funds. As more users join, the network becomes prone to more fragmentation of collateral. Given e.g. 1 Million users, and an average of 10'000 USD of transaction volume towards these users (over a given time span), a $\mathcal{P}$ would be required to lock up 10 Billion USD worth of collateral. However, in NOCUST, the intermediary does not need to lock up such insurmountable collateral to provide finality within two eons.

5.1.2 Entry Barrier. One other valuable addition in terms of usability is the ability by a $\mathcal{P}_i$ to enter into a $\not{\subset}$ without the need to broadcast any messages to $\mathcal{BC}$ and immediately start receiving payments, unlike in payment channel networks. This possibility of easy entry into a NOCUST system enables participants to join any number of hubs that they want, without having to incur any costs.

5.1.3 Potential Throughput. The throughput in terms of off-chain transactions per second achievable within a NOCUST system is only limited by the implementations of the $O_{\not{\subset}}$ and $\mathcal{P}$ specifications, which may take the form of a distributed system capable of matching the performance of today's commercial custodian payment solutions.

5.2 Risk

In this section we briefly present some of the important operational risks that come with running, and using, an off-chain payment solution that NOCUST mitigates.

5.2.1 Channel Monitoring. The main risk associated with the participants of an off-chain solution is the requirement that users must always stay online to monitor their payment channels, or outsource this operation, to guarantee that the channel does not successfully terminate with an outdated state. A $\mathcal{P}_i$ in NOCUST need only come online once per *eon* at least to audit its account or issue the appropriate challenges. This mitigates the associated cost with maintaining watch of a $\mathcal{P}_i$'s balance.

5.2.2 Transfer Delivery. To guarantee, or emulate, faster transaction finality, the hub may put up collateral that can be claimed by the recipients in case of failure within two eons. Through leveraging the same commitment scheme in Section 3.4.1 used to enable the delayed delivery mechanism, the hub can allocate collateral in bulk that can be claimed only by each recipient in case the delayed delivery system fails. Recipients then know an upper-bound to the money they are guaranteed to receive, even in case of failure, and can then calculate the risk involved with accepting payments, even when the total amount in the backlog of the delayed delivery system exceeds the collateral. This amount need only cover operation within two eons. This takes the rigid requirements of payment channels, and relaxes them such that liabilities can be properly negotiated in a granular manner, such that even the delivery guarantees of payment channel networks can be reached, while still eliminating the inconveniences of shuffling around and consolidating locked collateral, and keeping control of funds within the hands of the users.

5.2.3 Ledger Integrity. An intermediary managing a set of payment channels with collateral in them may suffer a devastating attack, due to natural disaster, nuclear warfare or otherwise, that would leave the operator paralyzed and/or suffering from data loss. This leads to an interesting situation whereby the intermediary may not be able to terminate its channels to recover funds properly due to its data loss, or rightfully reclaim some of the collateral that it is owed in a channel, as the counter-parties of $\mathcal{P}$ may possess more up to date information. A malfunctioning NOCUST $O_{\not{\subset}}$ need not suffer these financial losses as the $\mathcal{V}_{\not{\subset}}$ guarantees a commitment to the contents of $\mathcal{B}^L$ in $e-2$ regardless of the state of $O_{\not{\subset}}$.

5.2.4 $\mathcal{BC}$ Congestion. Under a congested $\mathcal{BC}$, a bidding war for correct channel termination may occur between a set of participants in a payment channel based intermediary. This can be likened to what is described as a mass-exit. During operation, a NOCUST intermediary $O_{\not{\subset}}$ may not be able to retain ledger integrity by utilizing $\mathcal{V}_{\not{\subset}}$ while $\mathcal{BC}$ is congested. However, rather than a malicious $\mathcal{P}_i$ being allowed to double spend their balance in $\not{\subset}$, the failure safety mechanism of $\mathcal{V}_{\not{\subset}}$ will be triggered, voiding the withdrawal and rolling back to the last commitment.

6 FUTURE WORK

In this section we discuss future work we believe would contribute to the efficacy of the proposed solution in different ways.

Cross-Instance Operations. An interesting venue to explore would be a specification for NOCUST instances to communicate with each other such that a $\mathcal{P}_i$ of one $\not{\subset}$ instance could transfer funds to a recipient $\mathcal{P}_j$ of another $\not{\subset}$ instance securely. It would be interesting to consider how NOCUST instances could employ Two-Party channels to facilitate such operations, as such a hybridization of solutions might lead to an elegant solution that delegates the routing concerns from participants to $O_{\not{\subset}}$ operators.

Privacy Enhancements. The operations performed in NOCUST using the data structure presented in Section 3.3 employ only asymmetric cryptography to provide authentication. It would be interesting to see a similar mechanism that utilizes zero-knowledge proofs, such as in the Zcash [34] protocol, to provide the same security guarantees while improving privacy.

Exchange Operations. The specification in this work for NOCUST allows the bi-directional transfer of balances in a $\not{\subset}$ instance. The augmentation of the protocol to allow securely exchanging, or swapping, one form of balance for another would certainly be a very interesting future contribution on top of NOCUST.

Trusted Execution Environments. The only checks that $\mathcal{V}_{\not{\subset}}$ performs on the $\mathcal{T}_{\not{\subset}}$ are simple constraint verification. It would be interesting to explore augmenting $\mathcal{V}_{\not{\subset}}$ to accept only $\mathcal{T}_{\not{\subset}}$ commitments generated within a trusted execution environment.

7 CONCLUSION

In this work we presented a basis for establishing a non-custodial 2$^{\text{nd}}$-layer financial intermediary that can securely facilitate payments between participants in its off-chain network without reliance on a consensus mechanism as in side-chains, but rather on a practical challenge-response protocol that leverages a simple and reliable data-structure.

Our construction NOCUST features multiple novel properties for off blockchain payments: (i) users can join the payment hub without the need for a costly on-chain transaction, (ii) locked up collateral in the hub can be zero up to the transaction volume of a disputable time window (to achieve trustless operation), (iii) the hub's collateral can be effectively managed in bulk, significantly reducing the management costs of the operator, compared to a two-party payment channel hub. We've moreover shown how users of a NOCUST construction can securely maintain custody of their funds, even under the hub's adversarial behavior or unavailability.

NOCUST empowers the individual to become it's custodian. In the future, we envision users to hold significant off-chain funds, as these can be transferred faster and at a lower cost than regular on-chain transactions. Even in the case of blockchain congestion or excessive transaction fees, the users' funds liquidity is guaranteed.

REFERENCES

[1] Jimuta Naik. Beginning of the early banking industry in mesopotamia civilization from 8th century bce. 2014.

[2] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296. USENIX Association, 2016.

[3] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59. USENIX Association, 2016.

[4] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model, 2016.

[5] Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, and Prateek Saxena. Scp: A computationally-scalable byzantine consensus protocol for blockchains. *IACR Cryptology ePrint Archive*, 2015:1168, 2015.

[6] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[7] Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. Service-oriented sharding with aspen. *arXiv preprint arXiv:1611.06816*, 2016.

[8] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2015.

[9] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *arXiv preprint arXiv:1702.05812*, 2017.

[10] Raiden network. http://raiden.network/.

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

[12] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[13] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. 2017.

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[15] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[16] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.

[17] Cynthia Dwork and Moni Naor. *Pricing via Processing or Combatting Junk Mail*, pages 139–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[18] Adam Back. Hashcash - a denial of service counter-measure. Technical report, 2002.

[19] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[20] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[21] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[22] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. Technical report, Cryptology ePrint Archive, Report 2016/701, 2016.

[23] Bitcoinj. https://bitcoinj.github.io/working-with-micropayments.

[24] Jay Freeman Gustav Simonsson Stephen F. Bell Steven Waterhouse David Salamon, Brian J. Fox. Orchid: A fully distributed, anonymous proxy network incentivized through bandwidth mining. https://orchidprotocol.com/whitepaper.pdf.

[25] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. *Proceedings of NDSS 2017*, 2017.

[26] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. Technical report, IACR Cryptology ePrint Archive, 2017: 635, 2017.

[27] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.

[28] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 455–471. ACM, 2017.

[29] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489. ACM, 2017.

[30] The inevitability of privacy in lightning networks. https://www.kristovatlas.com/the-inevitability-of-privacy-in-lightning-networks/.

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

[32] Loi Luu, Yaron Velner, Jason Teutsch, and Prateek Saxena. Smart pool: Practical decentralized pooled mining. *IACR Cryptology ePrint Archive*, 2017:19, 2017.

[33] Ralph C. Merkle. *A Digital Signature Based on a Conventional Encryption Function*, pages 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[34] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.