# Interhead Hydra
# Two Heads are Better than One

Maxim Jourenko[1,2], Mario Larangeira[1,2], and Keisuke Tanaka[1]

[1] Department of Mathematical and Computing Sciences,
School of Computing,
Tokyo Institute of Technology.
Tokyo-to Meguro-ku Oookayama 2-12-1 W8-55, Japan.
{jourenko.m.ab@m, mario@c, keisuke@is}.titech.ac.jp
[2] Input Output Hong Kong.
{maxim.jourenko, mario.larangeira} @iohk.io
http://iohk.io

**Abstract.** Distributed ledger are maintained through consensus protocols executed by mutually distrustful parties. However, these consensus protocols have inherent limitations thus resulting in scalability issues of the ledger. Layer-2 protocols operate on *channels* and allow parties to interact with another without going through the consensus protocol albeit relying on its security as fall-back. Prominent Layer-2 protocols are payment channels for Bitcoin that allow two parties to exchange coins, State Channels for Ethereum that allow two parties to execute a state machine, and Hydra *heads* [FC'21] for Cardano which allows multiple parties execution of Constraint Emitting Machines (CEM). Channels can be concatenated into networks using techniques such as Hashed Timelocked Contracts to execute payments or virtual state channels as introduced by Dziembowski et al. [CCS'18] to execute state machines. These constructions allow interaction between two parties across a channel network, i.e. the two endpoints of a path of channels. This is realized by utilizing *intermediaries*, which are the parties on the channel path which are in-between both endpoints, who have to pay collateral to ensure security of the constructions. While these approaches can be used with Hydra, they cannot be trivially extended to allow execution of CEMs between an arbitrary amount of parties across different Hydra heads. This work addresses this gap by introducing the Interhead construction that allows for the iterative creation of virtual Hydra heads. Of independent interest, our construction is the first that (1) supports channels with an arbitrary amount of parties and (2) allows for collateral to be paid by multiple intermediaries which allows to share this burden and thus improves practicality.

**Keywords:** Blockchain, State Channel, Channel Network

## 1 Introduction

Decentralized ledger were first introduced by Nakamoto [20] with the blockchain technology. Cryptocurrencies that are based on this design enjoy a steadily increasing popularity since then. However, while a wider adaption of decentralized ledger shows the relevance of this technology, the existing implementations struggle to be scalable to the increased demand. Transactions, e.g. payments, on a decentralized ledger require it to be processed through a consensus mechanism, which are classified as Layer-1 protocols. The potential transaction throughput of a ledger is limited by its consensus protocol and increasing it is highly non-trivial [5]. Transaction issuer can pay fees to increase the priority under which their transaction is processed, skipping the line. This results in the creation of a market place where processing a transaction requires payment of an amount of fees that correlates with the demand for processing transactions. For instance, on 20th April 2021 the average cost of processing a transaction in Bitcoin peaked at more than 60$ [3].

Layer-2 protocols are a classification of techniques that aim to reduce the number of transactions that have to be issued on a ledger by means of a layer of indirection. Layer-2 solutions include sidechains [10,24], Ethereum's Plasma [23], payment channels [22,21,6] and more generally state channels [7,8] notably Hydra heads [4]. This work focuses on state channels. Payment channels are setup by two parties using a transaction that locks their coins into a shared wallet. The channel stores a state which is how the funds that are locked inside it distributed between both parties. The parties can then perform an *offchain* protocol change this state. Lastly the parties issue one transaction to unlock the coins from the channel corresponding to its latest state. Note that the latter transaction is created first to avoid that coins are locked in the channel indefinitely if one of the parties is unresponsive. This way parties can perform $\mathcal{O}(n)$, $n \in \mathbb{N}$, payments with each other while only issuing $\mathcal{O}(1)$ transactions on the ledger thus improving the system's scalability. State channel extend this notion by allowing for the storage of arbitrary state which, with the conjunction of a sufficiently expressive scripting mechanism, can allow the two parties to execute state machines offchain. Hydra generalizes the notion of channels further by allowing an arbitrary amount of parties to participate in one channel. These parties can execute state machines, modelled in form of Constraint Emitting Machines (CEMs) [2], offchain. Further offchain protocols like Hash Timelocked Contracts (HTLCs) [22] and virtual channels [8,12,13] allow adjacent channels, i.e. multiple channels with one common party, to be concatenated into channel networks which allows parties to interact with another without the need of opening a new channel, reusing the existing channel infrastructure.

Whereas protocols for performing payments across a payment channel network, or execution of state machines across a state channel network exist, there is no analogous protocol for Hydra. Although HTLCs can be reused to perform offchain payments across Hydra heads, there is no way to execute arbitrary

[3] https://ycharts.com/indicators/bitcoin_average_transaction_fee

CEMs offchain across multiple, partially overlapping, Hydra heads. Moreover, the existing approach to connect state channels by using a virtual channel construction by Dziembowski et al. [8] does not seem to be easily extendable to channel constructions for more than two parties, as in Hydra, since it requires to uniquely identify malicious parties to subsequently punish them. For one, it is unclear whether we can uniquely identify misbehaviour in the case of channels with more than two parties. For another, the punish mechanisms in virtual channel constructions [7,8,12] use the ability to uniquely attribute fault to have the deviating parties all of their coins or collateral within the virtual channel to the honest parties. However, how a punish mechanism would look like for channels with multiple parties is unclear. Imagine a situation where all parties within a virtual channel play a game where all parties pay an amount of coins into the game and the winner will receive all of it, however, the winner is not decided upon until the end of the game. Then a party misbehaves to trigger the punish mechanism before the game resolves. A punish mechanism similar to those in the related work would result in termination of the game and payout of coins to all honest parties to ensure none of them lose any coins due to this. However, since the game has not yet been resolved, it is unclear how coins should be distributed among the honest parties in a fair manner.

*Our Contributions.* This work proposes a protocol to create *virtual* state channels and provides a security analysis. Our approach consists of two components, an Interhead state machine as well as a protocol that defines all parties' behaviour. The construction (1) can be executed iteratively to form a virtual channel across a channel network, (2) operates optimistically offchain, meaning, if all parties collaborate no transactions are added to the ledger, (3) is secure in the presence of a malicious adversary who can statically corrupt all but one party at the beginning of the protocol, (4) does not put honest party at risk of loss of their coins, (5) supports channels with an arbitrary amount of parties (6) allows for the presence of multiple intermediaries who can share the burden of committing required collateral, significantly improving practicality.

While this work focuses on the creation of a virtual channel for Hydra heads, we argue that the state machine proposed in this work can be implemented for other channel constructions on blockchains with sufficiently expressive smart contracts such as Ethereum. As of such we present the, to our knowledge, first virtual channel construction with properties (5) and (6) from above which we think is of independent interest.

*Structure.* In the remainder of this work, first we provide an overview of related work in Section 2 and provide background information relevant to this work in Section 3. Next, we introduce our approach. We present the state machine in Section 4 followed by how it can be implemented as a CEM in Section 5 and a protocol describing all parties' behaviour in Section 6. This is followed up by security analysis in Section 7. Lastly, we conclude in Section 8.

## 2 Related Work

*Channels.* Conceptually channels work as follows. Two parties open a payment channel between them by committing a *funding* transaction to the ledger that locks their funds into a shared wallet. A second *refund* transaction is setup that spends the coins within that wallet and refunds it to both parties, corresponding to how many coins were initially paid into the channel by each party. Note that the *refund* transaction is to be completed before the *funding* transaction to avoid that coins are locked within the channel indefinitely. The parties withhold committing the refund transaction to the ledger, but keep it in memory instead. The refund transaction represents the channel's state which is the result of committing the refund transaction to the ledger. A state transition occurs by creating a new refund transaction and implicitly or explicitly invalidating all previous refund transactions. Existing approaches differ in how invalidation is realized. Channels in the Lightning network [22] have both parties exchange invalidation keys that allow them to claim all coins within a channel if their counterparty publishes an old refund transaction. Duplex Payment channels [6] utilize timelocks, whereas Eltoo [21] utilizes hanging transactions, i.e. transactions where inputs can be changed after creation before committing them to the ledger. The notion of channels can be extended to state channels [19,7,8] by utilizing smart contracts. Such channels can store arbitrary state and, moreover, allow parties to execute state machines.

*Channel Networks.* Several protocols exist that allow to perform payments across a path of payment channels of length $n \in \mathbb{N}$. A payment within a network is emulated by performing it on each hop on the payment path. The challenge is to perform these payments atomically, i.e. it is performed on all channels within the payment path or on none. Here we distinguish between sender and recipient of a payment and the intermediary parties within a path of channels between both parties. Hash Timelocked Contracts (HTLCs) [22] make the recipient of a payment sample a secret $x \in \mathbb{N}$ s.t. $\mathcal{H}(x) = y$ where $\mathcal{H}$ is a cryptographic hash function. Each channel on the payment path sets up a conditional payment that performs the payment if the payee can tell the preimage of $y$ to the payer. Upon setup, the recipient discloses $x$ to its predecessor on the payment path allowing both parties to resolve the payment on their channel. In turn the predecessor learns $x$ and can forward it to reclaim its coins. A drawback of HTLCs is that the total required collateral, i.e. the amount of coins locked within conditional payments multiplied by the duration, sums up to $\mathcal{O}(n^2)$. Following approaches are Atomic Multi-Channel Updates [9] that reduces the total collateral to $\mathcal{O}(n)$ albeit it has been found that honest parties are vulnerable to loss of coins [13]. Furthermore, Jourenko et al. [13] introduce Payment Trees, a protocol that reduces the total collateral to $\mathcal{O}(n)$ but requires any party to commit up to $\mathcal{O}(\log n)$ transactions to the ledger in case of dispute, in contrast to $\mathcal{O}(1)$ transactions as with previous approaches. Sprites [19] allows for payments with total collateral in $\mathcal{O}(n)$ and without increasing the number of transactions above $\mathcal{O}(1)$ per party, but requires the use of a PreimageManager smart contract.

*Virtual Channels.* Alternative approaches to allow parties to interact with another across a network of channels are in form of *virtual channels*. Assume Alice and Ingrid, as well as Ingrid and Bob share a channel. These protocols allow two adjacent channels with one common party called the *intermediary* to create a third channel. In the above example Ingrid can act as intermediary to create a channel between Alice and Bob. These channels are created optimistic offchain, i.e. without committing any transaction on the ledger except in the case of dispute. Dziembowski et al. [7,8] created virtual state channels based on smart contracts, whereas lightweight virtual payment channels [12] allow for creation of virtual payment channel without the requirement of smart contracts.

*Hydra Heads.* The Cardano/Ouroboros Blockchain allows the execution of Constraint Emitting Machines (CEMs) [2] which are a form of state machines derived from Mealy Automata. Hydra [4] proposes a CEM and protocol that allows for the creation of multi-party state channels. Hydra is isomorphic, s.t. it allows parties to not only lock coins inside the channel, but a proper subset of the ledgers state. Doing so, the parties can interact with each other within a Hydra head the same way they could on a ledger. Due to this, simple payments can be performed across multiple Hydra heads by means of the HTLC protocol. However, whereas virtual state channels [7,8] allow parties to execute state machines across a state channel network, there are no methods to execute CEMs across a network of Hydra heads. This work fills this gap.

## 3 Background

*Notation.* In this work we make frequent use of tuples to structure data. Let $\alpha$ be an instance of a tuple of type $\mathcal{A}$ of form $(\alpha_0, \ldots, \alpha_n)$, $n \in \mathbb{N}$ where $\alpha_0, \ldots, \alpha_n$ are the entries' labels. Then we address entry $i \in \mathbb{N}, 0 \leq i \leq n$ of $\alpha$ using its name and the entry's label, i.e. $\alpha.\alpha_i$. Moreover we denote $\mathbb{N}$ as the set of natural numbers and $\mathbb{B}$ as the set of Boolean.

*Signature Schemes.* We assume the existence of two secure digital signature schemes as follows. We assume that both fulfill notions of **completeness** and **unforgeability**, however, we remain rather informal in the remainder. First, we assume a signature scheme [1] consisting of algorithms ($\mathsf{key\_gen}$, $\mathsf{verify}$, $\mathsf{sign}$) s.t. $\mathsf{key\_gen}(1^\lambda) = (vk, sk)$ creates a pair of secret key $sk$ and verification key $vk$ under security parameter $\lambda$, $\mathsf{sign}(sk, m) = \sigma$ creates a signature s.t. $\mathsf{verify}(vk, m, \sigma')$ evaluates to $\mathsf{True}$ if and only if $\sigma' = \sigma$. Second, we assume the existende of a multi-signature scheme [11,18] of form ($\mathsf{ms_setup}$, $\mathsf{ms\_key\_gen}$, $\mathsf{ms\_agg\_vk}$, $\mathsf{ms\_sign}$, $\mathsf{ms\_agg\_sign}$, $\mathsf{ms\_verify}$ ) where $\mathsf{ms\_setup}(1^{\lambda'}) = \Pi$ creates public parameter $\Pi$ with security parameter $\lambda'$, $\mathsf{ms\_key\_gen}(\Pi) = (\mathsf{vk}', \mathsf{sk}')$ creates a key pair consisting of secret key $sk$ and verification key $vk$', $\mathsf{ms\_agg\_vk}(\Pi, V) = \mathsf{avk}$ aggregates a set of verification keys $V$ into an aggregate verification key $\mathsf{avk}$, $\mathsf{ms\_sign}(\Pi, \mathsf{sk}', m') = \sigma''$ creates a signature $\sigma''$ of message $m$ corresponding to secret key $\mathsf{sk}'$ whereas $\mathsf{ms\_agg\_sign}(\Pi, V, S, m') = \sigma_{agg}$ aggregate a set of signatures $S$ into

aggregate signature $\sigma_{agg}$ s.t. $\mathsf{ms\_verify}(\Pi, m', avk, \sigma'')$ evaluates to $\mathsf{True}$ if and only if $\sigma'' = \sigma_{agg}$ and evaluates to $\mathsf{False}$ otherwise.

*The EUTxO Model.* Extended Unspent Transaction Outputs (EUTxO) were introduced by Chakravarty et al. [2]. EUTxO improve on the UTxO paradigm as used with ledgers such as Bitcoin introduced by Nakamoto [20] by allowing for the execution of smart contracts defined as Constraint Emitting Machines (CEMs) on the ledger, thus improving the system's expressiveness. An EUTxO based Ledger consists of a set of $(\mathsf{out}_{ref}, u)$ where $u$ is a EUTxO representing coins that are in circulation, and $\mathsf{out}_{ref}$ is a unique identifier that can be used to reference $u$ and is commonly derived from the context it was created in. A transaction $tx$ is a tuple of form $(I, O, r, S)$ where $I$ is a set of inputs, i.e. entries of form $(\mathsf{out}_{ref}, u)$, $O$ is a list of outputs, i.e. newly defined EUTxO, $r$ is a validity interval, i.e. $[r_0, r_1]$ where $r_0$, $r_1 \in \mathbb{N}$ are points in time, and $S$ is a set of signatures. If a transaction is sent to the ledger within the interval $r$, the amount of coins in $O$ is at least as large as the amount of coins referenced in $I$ and all validity scripts evaluate to $\mathsf{True}$, the transaction induces a state transition on the ledger by removing all entries in $I$ from its state and adding the newly defined EUTxO in $O$ to the ledger's state. A transaction is processed by the ledger within time $\Delta \in \mathbb{N}$. A EUTxO $u$ itself is a tuple of form ($\nu$, $value$, $\delta$) where $\nu \in \{0,1\}^*$ is a validator script written in a Turing complete language, $value \in \mathbb{N}$ is an amount of coins, and $\delta \in \{0,1\}^*$ is arbitrary data. An EUTxO can be spent making its coins accessible, if a party can show a redeemer value $\rho \in \{0,1\}^*$ s.t. $\nu(\rho, \delta, \sigma) = \mathsf{True}$, where $\sigma$ is the validation context that includes information on the transaction that spends $u$ as well as all EUTxO referenced in its inputs.

*Optimistic Offchain Protocols.* A class of protocols that improves the scalability of a ledger by minimizing the amount of transactions that are added to the ledger are offchain protocols. These protocols operate on structures called channels [22,21,6,8] and allow two parties to interact with each other, e.g. perform payments, without adding any transactions on the ledger in the *optimistic* case, i.e. when all parties collaborate and there is no dispute.

*Hydra Heads.* The Hydra CEM [4] allows for an arbitrary number of participants to move their EUTxO into a Hydra head and use them to interact with each other offchain. Hydra represents a channel structure between an arbitrary amount of parties. While operational, the Hydra head is in the state ($\mathsf{open}$, $K_{agg}$, $\eta$, $h_{MT}$, $n$, $T$) where $\mathsf{open}$ is the state's label, $K_{agg}$ is an aggregate verification key between $n$ participants, $\eta$ is the set of EUTxO that is hold off-chain, $h_{MT}$ is the root of a Merkel-Tree representing all participants and $T$ is the *minimum* duration of a contestation period that can occur when closing the Hydra head. The set of EUTxO that were moved to a head are stored within a snapshot which represents the set of EUTxO that are moved to the ledger upon closure of the head. Participants can modify this set of EUTxO by creating a new snapshot that supersedes previously created snapshots. Thus, the latest snapshot represents the

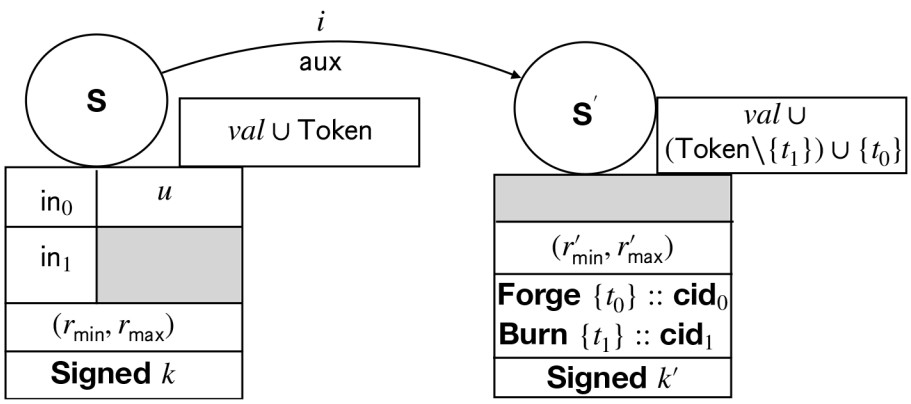


Fig. 1: Illustration of State Transition $S \to S'$ on input $(i, aux)$. The box below a state displays information on the transaction constraints for the transaction that performed the state transition. The box to the right of the state displays an overview of the *value* field of the EUTxO that represents the state. Transaction fields that are empty or implicit from context are grayed out or omitted for simplification.

state of EUTxO within the head that can be *enforced* on the ledger. Participants can create new snapshots while the head is in the open state. The head is closed by moving its state first to the closed and subsequently to the newestSN and final states. Lastly, a split transaction makes the EUTxO available on the ledger. Moreover, EUTxO can be incrementally decommitted which makes it available on the ledger without closing the head. This operation requires consent of all head members.

*Enforceable State.* Channels enable maintaining and modifying an *enforceable state*. For instance, in the case of payment channels the enforceable state is how a fixed amount of coins are distributed between two parties. In the case of Hydra it is a set of EUTxO defined by the most recent snapshot.

In the following we give an overview of relevant concepts.

*EUTxO*$_{\mathsf{MA}}$. The EUTxO model is extended by EUTxO$_{\mathsf{MA}}$ [3] to add multi-assets support. A EUTxO$_{\mathsf{MA}}$ is defined as a EUTxO but allows the *value* field to carry non-fungible tokens in addition to fungible coins. Moreover a transaction in the EUTxO$_{\mathsf{MA}}$ model has two more entries, i.e. it is of form $(I, O, \mathsf{forge}, \mathsf{fpss}, S)$ where forge is a token bundle that can define a positive amount of token in case they are minted, or a negative amount of token in case they are burned. Moreover, fpss is a forging policy script taking the validation context $\sigma$ as input and evaluates whether the transaction including its forge field is admissible. In the remainder we assume EUTxO$_{MA}$, however we continue using the term EUTxO for brevity.

*Constraint Emitting Machines.* Chakravarty et al. [2] showed a weak bi-simulation between programs running on the EUTxO ledger and Constraint Emitting Machines (CEMs) derived from Mealy machines [17]. Thus, CEMs can be used to define applications for an EUTxO ledger. A CEM is a tuple (**S**, **I**, **step**, **initial**, **final**) where **S** is a possibly infinite set of states, **I** is a set of input symbols, **initial**, **final** are functions $\mathbf{S} \to \mathbb{B}$ indicating initial and final states respectively and function $\mathbf{step} : \mathbf{S} \to \mathbf{I} \to \mathbf{Maybe}(\ \mathbf{S}, \mathbf{TxConstraints}\ )$ is a partial function that maps to a new state with constraints **TxConstraints**.

A CEM is implemented on the ledger as follows. A state **S** is represented by a EUTxO $u$ where validator $u.nu = \nu_S$ is unique to the state and enforces correctness of state transitions and transaction constraints **TxConstraints**. More abstractly, an Onchain Verification Algorithm (OVC) corresponding to the state is implemented and enforced on the ledger through $\eta_S$. Figure 1 displays how we illustrate states and transitions in this work. The boxes under each state **S** and $\mathbf{S}'$ display information on **TxConstraints** of the transaction used to perform the transition. For instance, in Figure 1 $S'$ requires that a token with currency identifier $\mathbf{cid}_1$ is burned and a token with identifier $\mathbf{cid}_0$ is forged. Moreover, it requires that the transaction has a validity interval of $[r'_{\mathsf{min}}, r'_{\mathsf{max}}]$, and that it contains a signature corresponding to verification key $k'$. Moreover it holds that the redeemer that spends the EUTxO representing state $S$ equals $\rho = i||\mathsf{aux}$. The EUTxO representing the two states $S$ and $S'$ within inputs and outputs of the states are implicit and omitted for simplicity.

*Thread Token.* A design pattern that allows to enforce that a given CEM (1) started in a valid initial state and (2) is unique compared to other instances of similar CEMs is using thread token. The validator of an initial state requires creation of a thread token with respective forging policy script. The token will be kept in all EUTxO value fields through a run of the CEM until it reaches a final state in which it is forced to be burned.

## 4 Overview

This section first introduces the setting and the concepts relevant to this work. Then, we present the Interhead state machine, which we implement as a smart contract in form of a CEM in Appendix 5. The protocol which describes how parties interact with another and the state machine is described in Section 6.

*Approach and Terminology.* Our terminology is chosen to be in-line with Hydra [4]. The aim of this work is to create the *Interhead construction* which consists of two parts. For one, the *Interhead state machine* is operated between two separate *Hydra heads*. For another, the *Interhead protocol* defines the behaviour of the involved parties depending on their role within the construction. The Interhead state machine operates across two heads and thus is split into two initially disjunct parts. These *partial state machines* are operated in parallel by the members of the respective heads resembling multi-*threaded* execution.

However, they are setup such that the threads can merge to open a regular Hydra head on the ledger in case of dispute. Similar to how a Hydra head maintains a set of EUTxO as *enforceable state* between its participants, the Interhead is used to create a *virtual head* that maintains a set of EUTxO as enforceable between a subset of participants of two Hydra heads. However, the head is created without opening a Hydra head on the ledger, thus it is *virtual*. While the correctness of the Interhead state machine is verified and ensured by the ledger, the behaviour of the parties themselves is defined by the *Interhead protocol*.

### 4.1 The Setting

We assume the existence of two Hydra heads, $H_b$, $b \in \{0, 1\}$. Out of each head, $n_b \leq H_b.n$ *participants*, which are a subset of parties, want to move part of their EUTxO within the respective head into a new *virtual* Hydra head $H^v$ thus enabling interaction between these participants. Moreover, there are $1 \leq n_i \leq \min(H_0.n, H_1.n)$ parties who are present in both heads and act as *intermediaries*. For simplicity, in the remainder of this work each party has exactly one role, i.e. intermediary or participant, however, note that parties can have both or neither role. We consider four sets of parties. For one, there are *participants* $G_b$ from either head. The union of these two sets is the set of the virtual Hydra head participants $G^v = G_0 \cup G_1$. Lastly, there is the set of *intermediaries* $G_i$.

*The Communication Model and Time.* We assume that communication between the parties happens through authenticated channels and is done within rounds s.t. a message sent at any round will be available to the recipients at the beginning of the following round. We assume there is a relation between a given communication round and the clock time [14,15,16] at which it is happening s.t. we use time and communication rounds interchangeably in the remainder.

*The Adversarial Model.* Our adversarial model is in line with related work. That is, we assume a malicious adversary who at the beginning of the protocol can statically corrupt all but one, i.e. up to $n_0 + n_1 + n_i - 1$ parties. Upon corruption the internal state of a party is leaked to the adversary and all communication to and from the party goes through the adversary. The adversary can make any corrupted party deviate from the protocol arbitrarily. Moreover, the adversary can reorder messages and delay them until the following communication round.

### 4.2 Desired Properties

The construction is designed to fulfill following properties. Note that as soon as the Interhead state machine transitions into the Hydra state space, the security properties of Hydra [4] hold. First, we define security properties and then give an overview of the challenges.

**Definition 1 (Collateral Liveness).** *If at least one intermediary is honest, all collateral is eventually available to them in an enforceable state.*

**Definition 2 (EUTxO Liveness).** *Eventually any honest party's EUTxO within the virtual head's enforceable state is available to them in an enforceable state (or the ledger) outside the virtual head or the Interhead state machine transitions into the Hydra state space.*

**Definition 3 (Balance Security).** *The sum of a honest party's coins is reduced only with their consent or the Interhead state machine transitions into the Hydra state space.*

*Security.* The construction must be secure for all honest participants. Even if all other participants of the Interhead construction are behaving maliciously, the honest party cannot lose any coins. This requires that the Interhead construction fulfils Balance Security, Collateral Liveness and EUTxO Liveness.

*Optimistic Offchain.* Our construction must be optimistic offchain, i.e. if all parties collaborate, a virtual Hydra head can be constructed, used, and closed without commitment of any transactions on the ledger.

*Multiple Intermediaries.* Our construction should allow for multiple intermediaries. While this does not provide any additional features to the construction itself, it is highly relevant for making the construction practical. The amount of collateral to be committed has to match the number of coins and tokens within the virtual head. Having multiple intermediaries allows us to split up the burden of committing sufficient collateral. However, doing this securely is in itself a highly non-trivial challenge. Existing approaches for virtual channels [12,7,8] all have only one intermediary that is in a position to provide security to the construction, but in turn can be uniquely blamed if they deviate from the protocol. The challenge of multiple intermediaries is to ensure that the group of intermediaries is able to provide security to the construction the same way as with one intermediary, however, honest intermediaries should not lose their collateral in case all remaining intermediaries behave maliciously.

*Constraints on Hydra Heads.* We require a few *minor* additional constraints to Hydra Heads to support virtual Hydra heads. The minimum amount of time it takes for a EUTxO to be available on the ledger – assuming head participants do not participate in an incremental decommit – depends on the contestation period $T$ of a head and sums up to $2T$, i.e. a minimum snapshot posting period $T_{SN} \geq T + R_C$ and a minimum hanging transaction posting period $T_{HT} \geq T + R_{SN}$[4] where $R_C$ and $R_{SN}$ are the duration of the validity intervals of transactions moving into the closed and newestSN states respectively. However, there is no upper bound for the time it takes to add a EUTxO on the ledger as parties might attempt to delay head closure by selecting large durations for $T_{SN}$, $T_{HT}$, $R_C$ and $R_{SN}$ respectively. We require upper bounds for these parameter, i.e. we require $T_{SN}^{max}$, $T_{HT}^{max}$, $R_C^{max}$ and $R_{SN}^{max}$ s.t. $T_{SN}^{max} \geq T_{SN} \geq T$, $T_{HT}^{max} \geq T_{HT} \geq T$, $R_C^{max} \geq R_C \geq \Delta$, $R_{SN}^{max} \geq R_{SN} \geq \Delta$ holds. Doing so does not prevent heads from closing which would impact Hydra's *liveness* property as participants remain free

to delay state transitions arbitrarily, i.e. it will never be too late to close a head. However, a party that attempts to close the head to make a EUTxO available on the ledger can assume that it will be available to do so after at most time $T^{max} \leq T_{SN}^{max} + T_{HT}^{max} + R_C^{max} + R_{SN}^{max} + 2\Delta$. This holds if it actively engages to do so, i.e. as long as it commits the required transactions to the ledger whenever possible. Note the additional $2\Delta$ represent the upper amount of time to perform the state transition to the *closed* state and committing the split transaction to the ledger.

*General Purpose Token.* Hydra heads are limited in that it is not possible to forge and burn arbitrary token. Special purpose token, such as thread token, can only be forged within a specific context specified within its forging policy script. A transaction forging such a token cannot be included in an enforceable state as this would result in the token being forged within a Hydra CEM state transition which would likely violate its forging policy script. One workaround of this is to adjust the token as well as the CEM it is used in to be aware of Hydra such that moving the CEM into and out of a Hydra head is permissible as well as the forging and burning of the specific token within a Hydra state. Another workaround is by means of the *generalized token* pattern. A generalized token is forged in an arbitrary context and as of such can be forged during any state transition of a Hydra head. Generalized token can be created either as fungible or non-fungible token. A CEM that makes use of token to perform functionality does not forge or burn the token, but instead takes the required amount of token as input when required, and releases the token in the CEM's final state at latest.

*Multi-Threaded CEM.* We extend the notion of thread token by allowing CEMs to hold multiple thread token. A CEM can spawn threads to be executed in parallel by having a transaction contain multiple EUTxO in its outputs, each representing a separate CEM state and holding at least one thread token. In turn, multiple threads can be merged into a single thread by having a transaction spending multiple EUTxO representing CEM states, consuming their thread token and defining one EUTxO in its output that contains all thread token in its value field. We use multithreading in two cases. For one, we use multiple threads – one thread per identity – to efficiently collect EUTxO that are to be moved to the virtual head. Note that this is similar to how the Hydra CEM collects EUTxO [4] to be moved into a head. Second, we initially spawn one thread in each head, i.e. each instance of a Interhead CEM has exactly two initial states containing one thread token each. If the Interhead resolves optimistically, the CEM remains separate and the threads are never merged. However, in case of a dispute or a lack of collaboration, when the Interhead is converted into a regular Hydra head, the threads will be merged.

*Compatibility to Hydra.* We strive to make our construction compatible to the existing work, i.e. Hydra heads, and we require at most minimal adjustments. Doing so we can rely on the groundwork done for Hydra heads and their security properties which allows us to focus on the Interhead construction.

### 4.3 The State Machine

Our approach is taking a Hydra state machine [4] and adapting concepts of UTxO based virtual channels [12,13] to create an Interhead state machine that maintains a virtual Hydra head. Our approach is inspired by Eltoo [21], i.e. we avoid punishing any participants by ensuring that the virtual Hydra head can always be opened on the ledger as a regular Hydra head in case of dispute.

The Interhead state machine is executed offchain in both Hydra heads $H_0$ and $H_1$. It maintains a set of EUTxO between identities $G^v$ as enforceable state. The Interhead can be resolved within both heads optimistically, but in case of a dispute it allows to enforce the enforceable state of the virtual head by opening a regular Hydra head with the same enforceable state on the ledger. Intermediaries are parties that must be participating in both heads, allowing them to ensure correctness of the system. Intermediaries allocate collateral into the system which is returned to them when the Interhead state machine resolves optimistically or when the virtual Hydra head is opened on the ledger. However, in the case in which all intermediaries are corrupted and fail their task, they will lose their collateral which in turn will be used to ensure enough coins are available to open the virtual head on the ledger.

*Time Phases and Assumptions.* We structure execution of the state machine into three phases, each operating under different assumptions and within disjunct time frames. (1) The *orderly* phase $T_O = [0, t_{C,\mathsf{start}})$ assumes that at least one intermediary is honest and all parties collaborate. (2) The *conversion* phase $T_C = [t_{C,\mathsf{start}}, t_{C,\mathsf{end}})$ assumes that at least one intermediary is honest. We require that this phase's duration is at least $\max(H_b.T^{\mathsf{max}}, H_{1-b}.T^{\mathsf{max}} + 2\Delta < t_{C,\mathsf{end}} - t_{C,\mathsf{start}}$. (3) The *punish* phase starting at $t_{C,\mathsf{end}}$ and going on indefinitely assumes that at least one party is honest which is ensured due to the adversarial model. If any of the assumptions during a phase is violated, the CEM escalates to the next phase by passage of time.

*Iterative Construction.* An Interhead maintains the same enforceable state as a Hydra head, but it is setup across two enforceable states instead of one. Note that as an Interhead itself maintains an isomorphic enforceable state, an Interhead can be setup iteratively across two Interheads or one Hydra head and one Interhead. This allows for Interhead constructions across multiple hops of an infrastructure of Hydra heads. In the following, for simplicity we assume that the Interhead is created across two Hydra heads that were opened on the ledger.

*Setup.* Each group $G_0$, $G_1$, $G^v$, $G_i$ jointly setups a multi-signature scheme as in Section 3 resulting in creation of aggregate signature keys $K_{agg,0}$, $K_{agg,1}$, $K_{agg}$, $K_{agg,i}$ respectively.

*The Interhead State Machine.* In the following we describe the states of the Interhead state machine as well as how state transitions are performed.

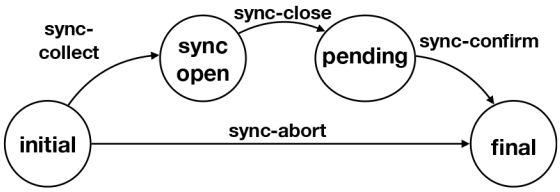


Fig. 2: State transitions in the orderly phase.

The Interhead state machine is structured using six states, namely initial, sync_open, pending, final, merged and punished. Here intial and final are the initial and the final states respectively. In addition, the Interhead state machine contains the states, input symbols, state transitions, and final states of the Hydra state machine as it performs a state transition into the *open* state of the Hydra state space in case of dispute or non-cooperation. For simplicity we abstract away from this and focus on the unique part of the Interhead state machine.

When starting the protocol the parties in each head setup their half of the Interhead state machine, starting in the Initial states respectively. The partial state machine that is operated within each head is displayed in Figure 2. Both partial state machines are operated in parallel, i.e. each Initial state spawns one thread of the overarching Interhead state machine. Intermediaries have to ensure that the initial states in both heads match s.t. in case of dispute the threads can be merged and the Interhead state machine can be transitioned into the Hydra statespace. For this reason the Initial states contain data that (1) ensures the threads can be merged and (2) the threads cannot be merged with a different Interhead state machine (3) the Hydra head maintains the same enforceable state as the Interhead.

State transitions within the state machine are limited to time phases, i.e. their validity interval must fall entirely within one of the phases. The transitions that can be performed within a phase are structured according to the purpose of that phase. In the following we present the Interhead state machine structured in these three phases, however, a complete overview of the whole state machine is illustrated in Appendix A.

*Orderly Phase.* Figure 2 illustrates states and transitions within the orderly phase. During this phase, the intermediaries have sole authority to perform state transitions. In fact, all transitions require a multi-signature corresponding to verification key $K_{\mathsf{agg},i}$ signed by all intermediaries. One instance of this partial state machine is executed within each head starting from the initial state. All transitions which require an input symbol with prefix sync are executed either on both heads, or on none. This is ensured through the synchronization protocol in Section 6 with one caveat: A corrupted intermediary can attempt to de-sync both partial state machines by acting in the last moment of the orderly phase and only providing their signature for one partial state machine. However, this can be detected by means of the pending state where intermediaries have to verify

and confirm that no de-sync attempt occured. Detection of a de-sync results in the state machine transitioning to the conversion phase.

There are multiple ways to prevent state transitions if one or all intermediaries are corrupted. For one, the validators within the state machine prevent incorrect state transitions within each partial state machine, however, this is not fully sufficient as we execute the state machine in parallel within two heads. Any honest intermediary can prevent a state transition that otherwise would result in a de-sync of both partial state machines by withholding their cooperation to create the required aggregate signature. Note the caveat mentioned earlier. Moreover, incorrect initialization and optimistic closure of the head can be prevented by any honest party again by withholding their cooperation for required aggregate signatures.

The state machine can reach two states from the initial state: (1) The sync-open state is reached through the sync-collect input. This step collects a set of EUTxOs $E_b$ from all participants, as well as a set of EUTxOs $C_b$ from intermediaries that acts as collateral. We require that all EUTxO contain only coins or fungible token. Although intermediaries are allowed to commit arbitrary amounts of collateral we require the constraint that the total amount of coins contained in $C_b$ has to be at least as much as the total amount of coins contained in $E_{1-b}$ and that that the amount and type of fungible token in $E_b$ and $C_b$ matches. As soon as the Interhead state machine reaches the sync-open state on both Hydra heads, the virtual Hydra head is opened and parties can modify its enforceable state. We omit displaying the collection of EUTxO in Figure 2 for simplicity, but details are shown in Section 5. (2) The final state can be reached using the sync-abort command aborting execution and releasing all previously committed EUTxO.

The pending state can be reached from sync-open through the sync-close input. This transition requires submission of a final-annotated partial snapshot of the virtual head. This final snapshot depends on the Hydra head, which it is submitted to, and contains the EUTxOs of its members as well as collateral from the intermediaries. The final snapshot has to be negotiated and confirmed between all parties, i.e. by $G^v$ and $G_i$ through multi-signatures corresponding to verification keys $K_{agg}$ and $K_{agg,i}$ respectively. Note that this step is similar to the concept of the optimistic head closure in Hydra. The purpose of the pending state is to detect attempts of corrupted intermediaries to de-sync the partial state machines' executed on each head. We discuss details in Section 6.

Lastly the final state can be reached through the sync-confirm input from the pending state, releasing EUTxO according to the negotiated snapshot. If the final state is reached within the orderly phase, the Interhead remains offchain and terminates. In the process, all EUTxOs previously committed by participants or committed as collateral by intermediaries are released.

Otherwise, if any party stalls execution, does not collaborate or a de-sync attempt is detected, no further transitions occur within the orderly phase and the state machine proceeds into the convert phase by passage of time.

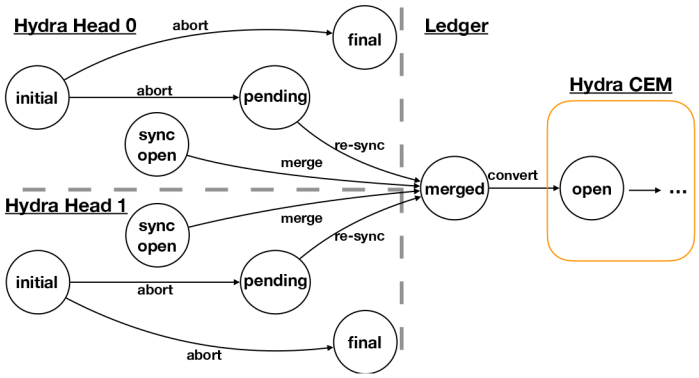


Fig. 3: State transitions in the conversion phase.

*Conversion Phase.* This phase is illustrated in Figure 3. Similar to the previous phase, all the intermediaries have sole authority to perform state transitions, however, now state transitions can be performed by any intermediary alone instead of requiring an aggregate signature. At this point the state machine has three outcomes. (1) If the head has not yet been opened and at least one party did not submit any EUTxO, then the Interhead can move directly from initial to final. (2) If no (honest) Intermediary performs any state transition, the state machine transitions into the punish phase by passage of time. (3) In all other cases, both partial state machines will be committed on the ledger and a Hydra head will be opened preserving the enforceable state.

A Hydra head is opened as follows. First, all Intermediaries enforce the state machine on the ledger which can be done through an incremental decommit of the state machine, or by closing the Hydra head. This requires time of up to $T^{\mathsf{max}}$. Then, if all parties committed EUTxO onto the ledger the state machine will proceed from the initial to the pending state as potentially a de-sync attempt was made. If both partial state machines reached either the pending or sync-open state, both threads are merged into one thread through the merged state using the merge input. The transaction that performs this transition returns the collateral to all intermediaries. Lastly, we transition from the merged state to the open state of a Hydra state machines through the convert input. Note that we remove any time restriction for the transition from the merged state to the open state.

*Punish Phase.* The last phase is illustrated in Figure 4. It is open-ended and any state transition can be performed by any party. It has two outcomes. If the state machine is still in the initial or pending state, the state machine can be safely aborted and transition into the final state. Otherwise, the state machine transitions into the open state of a Hydra state space. From the sync-open state the state machine transitions into the punished state with the punish input. The punished state is similar to the merged state with one exception. The intermedi-

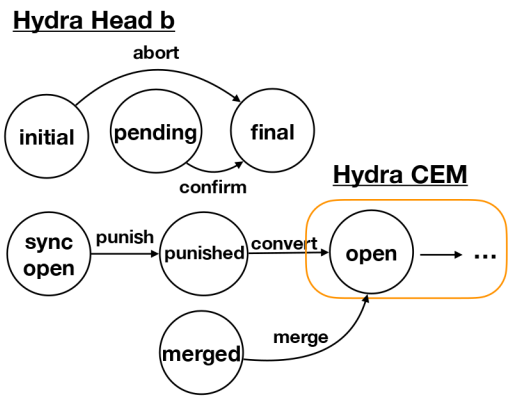


Fig. 4: State transitions in the punish phase.

aries collateral is not released. Instead, the collateral is used to provide enough coins to allow for the transition into Hydra's open state. Lastly, the state machine can transition from the merged state to Hydra's open state in this phase as well, in case this has not yet been done within the conversion phase.

## 5 CEM Construction

In the following we provide details of the CEM by describing each state as well as the constraints provided by each verifier. We illustrate the CEM using figures 6 - 14. Note that we do not illustrate all possible transitions as some are similar to another. Moreover, due to space constraints we do not formally define each verifier's behaviour but describe it on a high level only.

### 5.1 Parameter

The parameters under which the Interhead CEM is executed are negotiated among all participants and intermediaries in the beginning. We structure the data stored within the CEMs state in data $\delta^v$ that is required for opening the virtual head on the ledger, data $\delta_c$ that is common to the two partial CEMs in both $H_0$ and $H_1$ as well as data $\delta_b$ that is only relevant in each individual head $H_b$.

First, $\delta^v$ is a tuple of form ($K_{\mathsf{agg}}$, $\eta$, $h_{\mathsf{MT}}$, $n$, $T$, $\mathsf{cid}_0$, $\mathsf{cid}_1$) where the first four parameter are the virtual head's state and $\mathsf{cid}_b$ is the currency ID for thread token $t_{s,b}$ in head $H_b$. These parameter are not derived during execution of the CEM, but represent a commitment from the Interhead participants to create a virtual head with the respective parameters and thread token. For one, $\delta^v$ is used to ensure that always the same virtual head is created, even if only data from one partial CEM is available which is the case when the CEM enters the punish phase. The currency IDs are stored to ensure that the Interhead CEM instance

is unique and that both partial CEMs are tied to another, i.e. no partial CEM can be merged with another similar Interhead instance. Note that the EUTxO contained in $\eta$ are not allowed to contain unique non-fungible token.

Second, $\delta_{\mathsf{c}}$ consists of tuple $(K_{agg,i}, h_{MT,i}, n_i, T_o, T_c)$ which contains data on the intermediaries, i.e. their aggregate verification key $K_{agg,i}$, the head of the Merkel tree containing all individual verification keys $h_{MT,i}$ and the number of intermediaries $n_i$. Moreover it contains points in time $T_o$, $T_c \in \mathbb{N}$ specifying the end of the orderly and conversion time phases respectively. Verifier ensure that a transition happens within the orderly timezone by ensuring that for the transition's time frame $[r_{\mathsf{min}}, r_{\mathsf{max}}]$ holds that $r_{\mathsf{max}} \leq T_o$. Similarly verifier ensure that a transaction is within the conversion time frame by checking that $T_0 < r_{\mathsf{min}} < r_{\mathsf{max}} \leq T_c$. Lastly a transition happens in the punish phase if $r_{\mathsf{min}} > T_c$.

Lastly, $\delta_b$ consists of tuple $(b, K_{agg,b}, \eta_b, h_{MT,b}, n_b, \mathsf{col}_b)$ where $b \in \{0, 1\}$ is a bit identifying the order of both partial CEMs, $K_{agg,b}$ is an aggregate verification key, $\eta_b$ is a commitment of the EUTxO that the parties will move to the virtual head and it must hold that $\eta = \eta_0 \cup \eta_1$, $h_{MT,b}$ is the root of the Merkel tree consisting of the individual verification keys of $G_b$, $n_b = |G_b|$ is the number of participants joining from $H_b$, and $\mathsf{col}_b$ is the collateral required to be paid by the intermediaries. Note that $\mathsf{col}_b$ has to be at least as large as the amount of coins contained in the EUTxO in $\eta_{1-b}$. The quantity and type of fungible token submitted in the collateral has to match the number of token submitted by the participants.

### 5.2 The Orderly Phase

*The Initial State.* The initial state is created in both heads $H_b$ with parameter $\delta^v$, $\delta_c$, $\delta_b$. Each participant and intermediary verify that the parameter are as negotiated and, moreover, the intermediaries ensure that both initial states match and can be converted during the conversion phase in case of dispute. The transaction that sets up the initial state is signed using the aggregate verification keys of the participants in the respective head as well as the intermediaries. A party does only sign the transaction after positively verifying its correctness. The initial state requires that a state thread token, as well as participation tokens are provided in form of generalized token as input. The currency ID $\mathsf{cid}_b$ has to match the currency ID of the thread token that is provided as input. We require one participation token for each participant and each intermediary. The transition creates $n_b + n_i$ separate outputs, each containing one participation token and can be spent by exactly one participant and intermediary respectively.

*Commiting EUTxO.* Similar to Hydra heads, all participants and intermediaries commit a set of EUTxO each to the Interhead which is done in parallel. A commitment is displayed in Figure 5. Each participant and intermediary create one *commit* transaction that spends a participation token and several of their EUTxO within its inputs and stores information about them within its state $U_{i,b}$.

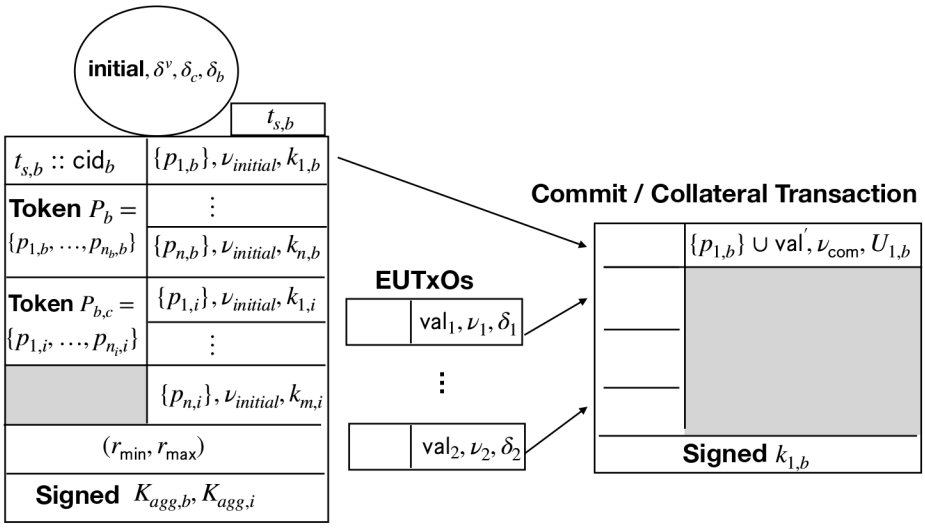


Fig. 5: Commitment of EUTxO and collateral into the Interhead.

*The Sync Open State.* If all parties created a commit transaction, they can be spent by the *sync-open* transaction as shown in Figure 6. The *sync-open* state verifies that the set of EUTxO $\eta = (U_{1,b}, \ldots, U_{n_b,b})$ committed by the participants, matches their original commitment, i.e. $\eta = \delta_b.\eta_b$. Moreover, it verifies that the collateral EUTxO $\eta_b^c = (U_{1,b}^c, \ldots, U_{n_i,b}^c)$ that are committed by the intermediaries contain a sufficient number of coins and fungible token. We store $\delta_b^{\mathsf{open}}$ in the CEMs state which equals $\delta_b$ but we replace $\delta_b.\mathsf{col}_b$ with $\eta_b^c$.

*Aborting.* Creation of the Interhead can be aborted by a transition from the initial state to the final state as shown in Figure 7. The final state makes the EUTxO that were committed available via Split transactions similar to Hydra [4].

*The Pending State.* The pending state can be reached from the sync-open and is shown in Figure 9. The pending state is used to give an opportunity for honest parties to either confirm head closure or to proceed to the conversion phase instead – in case a de-sync attempt was detected. The transition requires an as final annotated snapshot which transforms the EUTxO sets $\eta_b$ and $\eta_b^c$ into $\eta_b'$. The final snapshot $\eta_b'$ contains two components. For one, it contains a partition of the EUTxO within the whole Interhead namely the EUTxO that will be made in head $H_b$ upon closure. Moreover, it contains EUTxO that pay back the intermediaries' collateral. Note that the amount of coins that are paid back to the intermediaries within head $H_b$ might be less than what was paid by the intermediaries upon opening the head, for instance, if participants from $H_{1-b}$ performed payments to participants in head $H_b$. However, as the coins within the partial CEM is constant, the participants' coins will be taken from the coins

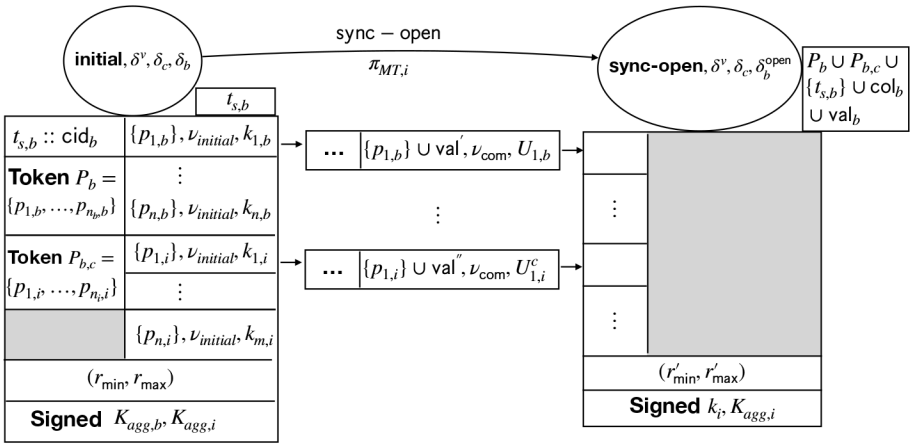


Fig. 6: Transition from initial to open state limited to the orderly phase.

submitted as collateral. However, in that case, this difference in coins will be available as additional collateral in $H_{1-b}$. Each intermediary has to ensure that the sum of collateral paid back to them in both heads is equal to the collateral they originally paid into creating the Interhead.

*The Final State.* In addition to aborting opening the Interhead, the final state can be reached from the pending state by a confirmation of the intermediaries as shown in Figure 10. Similarly to the abort case, the UTxO sets are made available within the transaction's outputs. However, this time the EUTxO that are made available are taken from the final snapshot $\eta'_b$.

*Making EUTxO available.* The final state partitions all EUTxO where each partition is spent by a Split transaction as shown in Figure 11 and is derived from the Hydra CEM. The Split transaction contains the EUTxO within the respective partition in its outputs.

### 5.3 The Conversion Phase

The conversion phase can conclude in two ways. For one, any intermediary can convert the Interhead CEM into a regular Hydra CEM. This requires that both partial CEMs are decommitted into a common enforceable state or the ledger respectively. This can happen by means of incremental decommits or closure of Hydra heads within time $T^{\text{max}}$. Note that no other state transitions are permitted within the Interhead CEM but conversion to a regular Hydra head. All transitions can be performed by an intermediary only, but now do not require a signature corresponding to the intermediaries' aggregate verification key $K_{\text{agg},i}$. For another, if the conversion has not been performed, in case no honest intermediary exists, the CEM transitions into the punish phase.

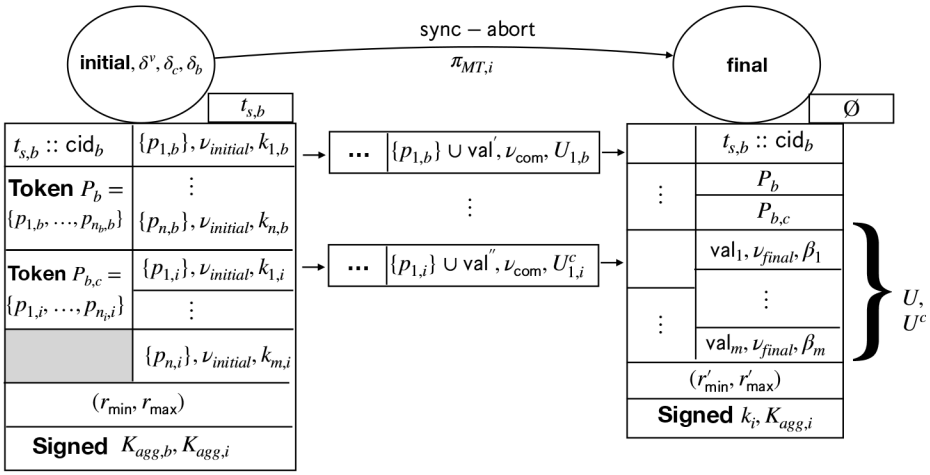


Fig. 7: Abortion of the construction during the orderly phase.

*Abort.* If there are any participants or intermediaries that have not yet committed EUTxO to the ledger, opening the Interhead can be aborted similarly to the way it is done during the orderly phase. However, if all participants and intermediaries committed EUTxO, aborting requires to transition into the pending state instead, because a de-sync attempt might have happened. Note that if the abort is permissible but the CEM transitioned into the pending state, it will be performed after during the punish phase.

*The Merged State.* Conversion to a regular CEM is by means of the *merged* state as shown in Figure 12. Both partial CEMs can be merged, either from the open-sync state or the analogous pending state and any combination of both. The merge state requires that two generalized thread token are present and match $\delta^v.\mathsf{cid}_0$ and $\delta^v.\mathsf{cid}_1$. This ensures that only the intended partial CEMs can be merged as both non-fungible thread token are unique. Moreover, it is verified that the EUTxO sets match the initial commitment, i.e. $\delta^v.\eta = \delta_0.\eta_0 \cup \delta_1.\eta_1$. This ensures that, in case all participants of one head as well as all intermediaries are corrupted, it is not possible for them to commit less EUTxO than required for the virtual Hydra head. The transaction releases all generalized token within its outputs, but similarly to the initial state it takes one thread token as well as $\delta^v.n$ participation token for all participants across both heads as input. Lastly, the CEM has one output for each set of EUTxO committed by the intermediaries to release their collateral in the same manner it is released in the final state. Note that from this point on, only information in $\delta^v$ is required and the remainder is removed from the state. As shown in Figure 13 any participant can perform a transition into the *open* state of a regular Hydra CEM. We do not limit this transition to the conversion phase s.t. it can be performed indefinitely after start

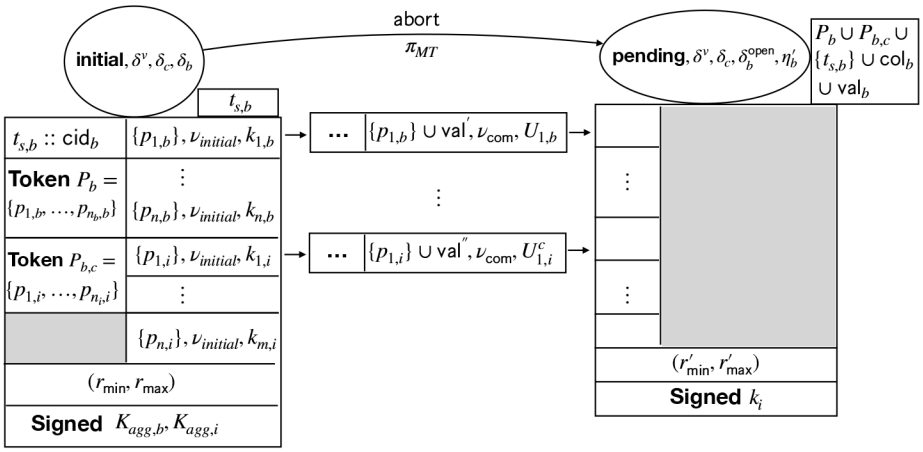


Fig. 8: Abortion during the conversion phase if all EUTxO and collaterals have been committed.

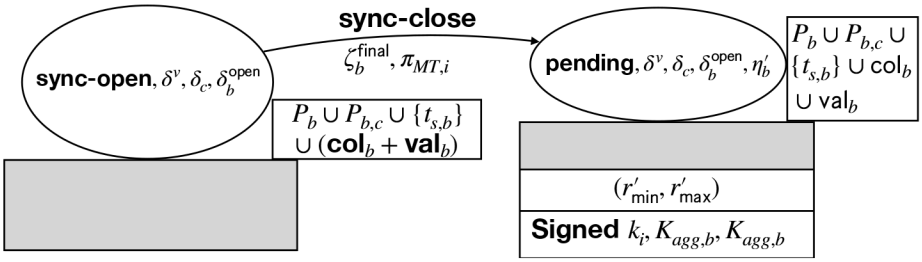


Fig. 9: First step of the optimistic closure during the orderly phase.

of the conversion phase. The state is directly derived from the data in $\delta^v$, i.e. $K_{\text{agg}} = \delta^v.K_{\text{agg}}$, $\eta = \delta^v.\eta$, $h_{\text{MT}} = \delta^v.h_{\text{MT}}$, $n = \delta^v.n$, $T = \delta^v.T$.

### 5.4 The Punish Phase

The purpose of the last phase is to allow any honest party to open the virtual Hydra head between all participants, even in the case that all other parties are corrupted. The Interhead CEM transitions into a Hydra CEM without the need of merging both partial CEMs. To ensure that this is possible, the coins and fungible token that are necessary for opening the Hydra head are taken from the collateral of the intermediaries who will lose it in the process. All transitions in this phase can be performed by any participant or intermediary.

*The Punished State.* If the CEM is in the sync-open state when entering the punish phase the CEM will transition into a regular Hydra CEM via the punish state as shown in Figure 14. This conversion can be performed within the same

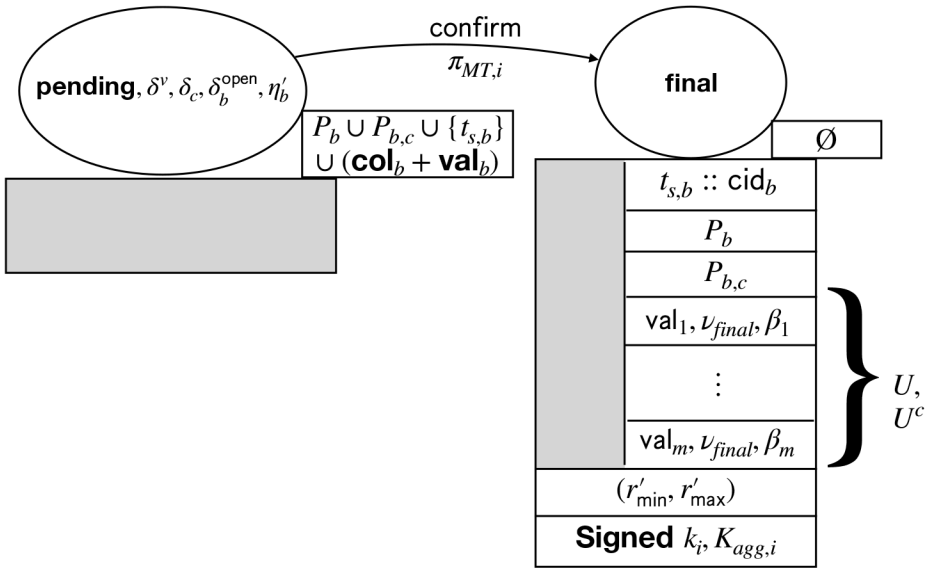


Fig. 10: Confirmation of head closure or abort. Requires aggregate signature only during orderly phase.

enforceable state in which the partial CEM is executed and thus requires no incremental decommit and neither closure of the Hydra head. The transitions from and to the punished state are similar to those to and from the merged state with the exception that we only verify correctness of the data from the local Hydra head, i.e. $\delta_b.\eta$ and $\delta^v$.cid. We re-use the existing thread token $t_{s,b}$ for the Hydra CEM. The collateral of the intermediaries is not made available through the transaction's outputs but used to finance conversion to the open state.

*Open Ends.* If the Partial CEM is in the initial state it can be aborted with a transition to the final state. Moreover, if the CEM was in the pending state it can now safely transition to the final state as no de-sync occured. Lastly, the transition from the merged state to the open state of a Hydra CEM is open ended and thus can be performed in the punish phase as well.

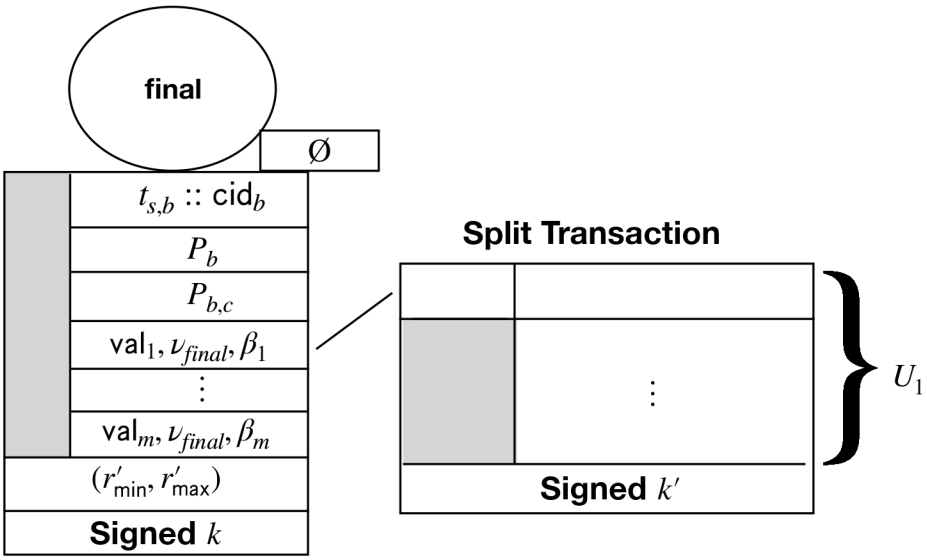


Fig. 11: Split transactions make all EUTxO and collateral available within their outputs.

## 6 The Protocols

We define the behaviour of honest parties in form of a protocol, split into three sub-protocols protocols. The initialization protocol is executed at the beginning, the *synchronization* protocol is executed by the intermediaries, and the *optimistic closure* protocol is executed for resolving the state machine within the orderly phase.

*Initialization Protocol.* At the beginning, the parameter of the initial state are negotiated between all participants and intermediaries, and the intermediaries need to verify that both initial states match and can be merged within the conversion phase. For this reason we require that the initial state contains aggregate signatures corresponding to both verification keys, $K_{\mathsf{agg}}$ and $K_{\mathsf{agg,i}}$. Any party provides their signature for this only in case of a positive verification of the relevant initial state.

*Synchronization Protocol.* While the state machine already enforces that all transitions within the orderly phase require consent from all intermediaries by requiring an aggregate signature corresponding to verification key $K_{agg,i}$, the synchronization protocol describes in which situation an intermediary provides their signature to approve a certain state transition. A state transition is approved by a party under two conditions. (1) All remaining inputs except the aggregate signature to perform the transition on both heads are known to the party. (2) If state transition sync-collect has been approved, then state transition

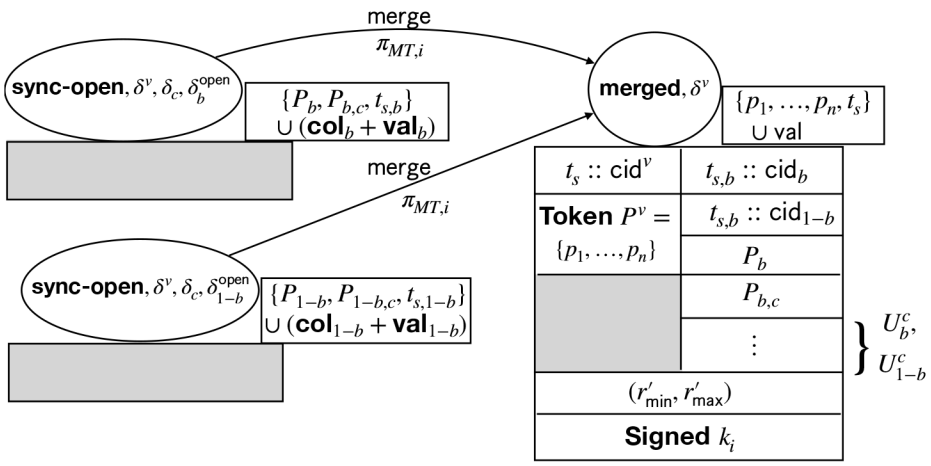


Fig. 12: Merging of both partial CEMs within the same enforceable state or the ledger. Transition can be performed from any combination of sync-open and pending states. Intermediaries' collateral is unlocked.

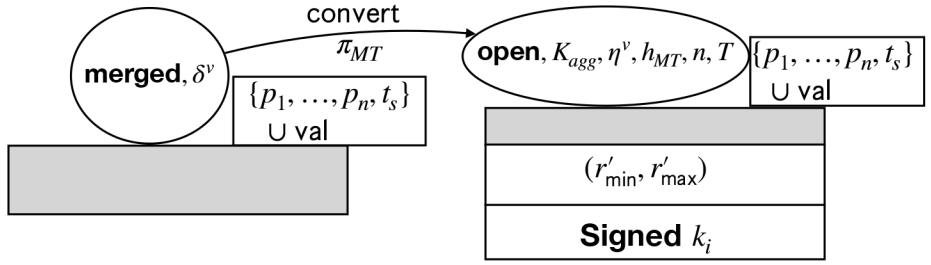


Fig. 13: Conversion into a regular Hydra head. The head's state can be inferred from $\delta^v$ directly. Transition originating from punished state is analogous.

sync-abort cannot be approved, and vice versa. Note that this implies that all parties have submitted EUTxO corresponding to the commitment $\delta^v.\eta$ and all intermediaries committed sufficient collateral. This is required to allow the Interhead to transition into the merged state if necessary. Note that the intermediaries create two aggregate signatures, one for each partial state machine.

This protocol allows that a state transition is performed either on both partial Interhead state machines or on none. However, there is one caveat. One corrupted intermediary can wait until its signatures were all that is remaining for the two aggregate signatures. Then they only complete one of the aggregate signatures. If this is done for sync-abort or sync-confirm transitions, this results in one head to close earlier than the other which is non-criticial as it does not impact security of the system. However, if it is done for sync-collect or sync-close transitions this could result in honest intermediaries to lose their collateral. Due to this, if this

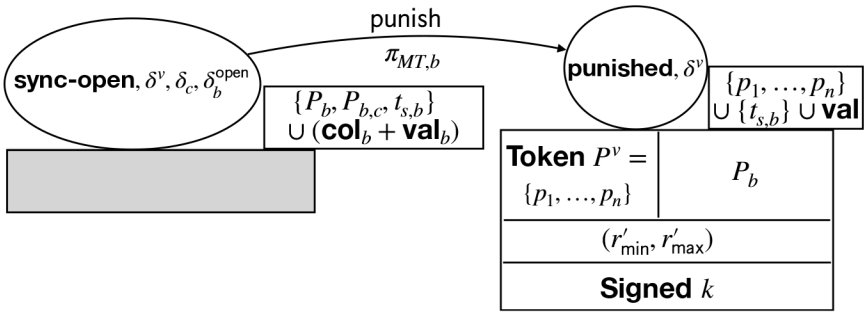


Fig. 14: Transition into punished state. Thread token is reused thus not provided as input. Transition into open state of Hydra CEM is analogous compared to originating from the merged state.

circumstance *can* occur, the state machine requires to proceed to the pending state. The pending state can be left through consent of all intermediaries through sync-confirm transition, however, if any intermediary detects this attempted desync, they will refrain from cooperating such that the CEM transitions into the conversion phase. Lastly, if the Interhead state machine proceeds into the conversion phase, each honest intermediary proceeds to actively commit the Interhead state machine on the ledger, potentially by closure of the Hydra heads, and perform state transitions until the merged state is reached.

*Optimistic Closure Protocol.* To close the Interhead state machine during the orderly phase we proceed similar to the optimistic head closure of Hydra. All parties negotiate two partial snapshots $\eta_0$, $\eta_1$ for both heads $H_0$ and $H_1$. These snapshots represent two partitions of all EUTxO within the Interhead's enforceable state. (1) $\eta_b$ contains those EUTxO that are to be released within head $H_b$ as well as (2) the collateral that will be paid out to the intermediaries within $H_b$. Upon negotiation, all parties provide aggregate signatures of $(\mathsf{final}, \eta_b)$ corresponding to verification keys $K_{\mathsf{agg},b}$ and $K_{\mathsf{agg},\mathsf{i}}$.

How the EUTxO are to be partitioned is free to the individual parties, however, there are a few things that have to be considered by each individual party. A participant who is member of head $H_b$ but not head $H_{1-b}$ should negotiate that all EUTxO that represent payments to itself must be in $\eta_b$. Similarly, EUTxO state represent state machines the party participates in should be in $\eta_b$ as well. This is due to if all of the parties in $H_{1-b}$ are corrupted, all EUTxO in $\eta_{1-b}$ are potentially lost. Moreover, each intermediary has to ensure that the sum of collateral it receives within both partitions is equal to the sum of collateral it paid into the CEM. Note that the amount of collateral paid out to the intermediaries will change within each head depending on how the EUTxOs are partitioned since the total amount of coins within the snapshot in $\eta_b$ cannot change from the amount of coins committed by participants and intermediaries

in $H_b$. However, the total amount of coins paid back to the intermediaries as released collateral will remain constant across both heads $H_0$ and $H_1$.

Any honest party must adhere to this behaviour during negotiation since otherwise the security properties in Section 4.2 cannot be guaranteed.

## 7 Security Proofs

In the following we provide security statements for the Interhead construction, composed of the Interhead state machine and the Interhead protocols. This corresponds to the desired properties in Section 4.2.

**Lemma 1 (Termination).** *If at least one party is honest, the Interhead CEM eventually reaches a final state or transitions into the Hydra CEM statespace.*

*Proof.* There are two cases. First, if all parties collaborate and do not behave maliciously, the Interhead CEM can reach a final state across two paths during the orderly phase, i.e. either through an abort, or through optimistic head closure. If this is not performed, the honest party waits until enough time passed for the CEM to enter the punish phase which will eventually happens. The state transitions within the punish phase have the only requirement a proof that the party is either intermediary or participant of the virtual head which can be performed by the honest party. There is a path of all states within the CEM's state space that either lead to a final state, or into the Hydra CEM state space.

**Theorem 1 (Collateral Liveness).** *If at least one intermediary is honest, the Interhead construction has the collateral liveness property.*

*Proof.* We note that there are two states that enforce that any collateral, that was committed prior, is made available within any enforceable state. These two states are the final state and the merge state. As shown in Lemma 1, eventually the Interhead CEM either reaches a final state or the Hydra CEM statespace. There is only one path a run through the Interhead CEM can take that does not end in a final state or contains the merge state. This is when the Hydra CEM has a run that contains the punished state.

Thus, what is left to show is that any honest intermediary $\mathcal{P}$ can enforce that a run does not contain the punished state. We observe that the punished state can only be reached from the sync-open state and only within the punish phase. In the following we assume that there are two partial CEMs, $I_b$, $b \in \mathbb{N}$ and CEM $I_b$ is in the sync-open state. In the following we reason about the potential states of partial CEM $I_{1-b}$.

First, $I_{1-b}$ can be in the initial state, or the pending state. Reaching the sync-open state from the initial state requires an aggregate signature of the group of intermediaries for which collaboration of $\mathcal{P}$ is required. As $\mathcal{P}$ is honest, they only collaborate with creation of the aggregate signature if all parties committed EUTxO consistent with commitment $\delta^v.\eta$ and all intermediaries committed sufficient collateral. Moreover, at the beginning of the protocol $\mathcal{P}$ confirmed that

both initial states of the partial Interheads are consistent, especially contain the same values for $\delta^v$. Since all parties committed transactions and collateral, the Interhead prevents a transition from the initial state to the final state before the punish phase. However, $\mathcal{P}$ can enforce a transition from the initial state to the pending state by providing proof they are an intermediary during the conversion phase.

Second, $I_{1-b}$ can be in the sync-open state and remain there, or transition into the pending state during the orderly phase. However, $I_{1-b}$ cannot reach the final state from the pending state during the orderly phase, as this requires an aggregate signature from the intermediaries which requires collaboration from $\mathcal{P}$ who only collaborates if both partial CEMs are in the pending state. Additionally, there is no transition from the pending state to the final state within the conversion phase. Thus, $I_b$ cannot transition into the final state from the pending state before the punish phase.

In summary, partial CEM $I_{1-b}$ is either in the pending or the sync-open state, or can be brought into the pending state by $\mathcal{P}$ during the conversion phase.

Thus, as soon as the Interhead CEM reaches the conversion phase, $\mathcal{P}$ closes both Hydra heads and performs the state transition of $I_{1-b}$ into the pending state if necessary. As both partial Interheads are within the same enforceable state $\mathcal{P}$ can perform the state transition into the merged state preventing any partial CEM to transition into the punished state. This requires at most time $t = \max(H_b.T^{\mathsf{max}},\ H_{1-b}.T^{\mathsf{max}} + 2\Delta$. As it holds that for the duration of the orderly phase $t_{C,\mathsf{end}} - t_{C,\mathsf{start}} > \max(H_b.T^{\mathsf{max}},\ H_{1-b}.T^{\mathsf{max}} + 2\Delta = t$, the honest intermediary $\mathcal{P}$ can perform transition into the merged state before the punish phase starts.

**Theorem 2 (EUTxO Liveness).** *If at least one party is honest, the Interhead construction has the EUTxO liveness property.*

*Proof.* Due to Lemma 1 we have two cases to consider for any honest party $\mathcal{P}$. Either the CEM reaches a final state, or it reaches the Hydra CEM state space. In the latter case, we are finished. In the former case we have two cases. For one, the CEM can abort which unlocks all previously committed EUTxO in which case we are finished. Otherwise, the final state is reached through the sync-open and pending states through an optimistic closure. In that case, the EUTxO that are unlocked depend on the negotiation during the optimistic closure protocol. $\mathcal{P}$'s collaboration of an aggregate multisignature is required to perform optimistic closure. If $\mathcal{P}$ is member in Head $H_b$ it verifies that all of the EUTxO it is related with are present in the snapshot partition $\eta_b$. If $\mathcal{P}$ is member in both Hydra heads it ensures all of the EUTxO it is related with are in either snapshot partition. In either case, all EUTxO within the partitions are made available within the Hydra heads $\mathcal{P}$ is member of.

**Theorem 3 (Balance Security).** *If at least one party is honest, the Interhead construction has the balance security property.*

*Proof.* This follows directly from Theorem 1 and Theorem 2.

## 8 Conclusion

In this work we present the Interhead construction, an approach to create virtual Hydra heads enabling communication that goes beyond simple payments but instead allows for the execution of arbitrary state machines between participants across a network of Hydra heads. We define security properties Collateral Liveness, EUTxO Liveness and Balance Security and prove them in the presence of a malicious adversary. We present the first virtual channel construction that supports channels with an arbitrary number of parties and that collateral is contributed by multiple intermediaries. Our construction thus closes the gap between Layer-2 protocols based on Hydra heads and Payment or State channels.

## References

1. Canetti, R.: Universally composable signature, certification, and authentication. In: Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004. pp. 219–233. IEEE (2004)
2. Chakravarty, M.M., Chapman, J., MacKenzie, K., Melkonian, O., Jones, M.P., Wadler, P.: The extended utxo model. In: 4th Workshop on Trusted Smart Contracts (2020)
3. Chakravarty, M.M., Chapman, J., MacKenzie, K., Melkonian, O., Müller, J., Jones, M.P., Vinogradova, P., Wadler, P.: Native custom tokens in the extended utxo model. In: International Symposium on Leveraging Applications of Formal Methods. pp. 89–111. Springer (2020)
4. Chakravarty, M.M., Coretti, S., Fitzi, M., Gazi, P., Kant, P., Kiayias, A., Russell, A.: Hydra: Fast isomorphic state channels. In: International Conference on Financial Cryptography and Data Security. Springer (2021)
5. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., et al.: On scaling decentralized blockchains. In: International Conference on Financial Cryptography and Data Security. pp. 106–125. Springer (2016)
6. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Symposium on Self-Stabilizing Systems. pp. 3–18. Springer (2015)
7. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: Perun: Virtual Payment Hubs over Cryptocurrencies. IEEE (2017)
8. Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 949–966. ACM (2018)
9. Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 801–815. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3345666
10. EthHub: Sidechains (2021), `https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/sidechains/`
11. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC Research & Development (71), 1–8 (1983)

12. Jourenko, M., Larangeira, M., Tanaka, K.: Lightweight virtual payment channels. Cryptology ePrint Archive, Report 2020/998 (2020), https://eprint.iacr.org/2020/998
13. Jourenko, M., Larangeira, M., Tanaka, K.: Payment trees: Low collateral payments for payment channel networks. In: International Conference on Financial Cryptography and Data Security. Springer (2021)
14. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Theory of Cryptography Conference. pp. 477–498. Springer (2013)
15. Kiayias, A., Litos, O.S.T.: A composable security treatment of the lightning network. IACR Cryptology ePrint Archive **2019**, 778 (2019)
16. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 705–734. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
17. Mealy, G.H.: A method for synthesizing sequential circuits. The Bell System Technical Journal **34**(5), 1045–1079 (1955)
18. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures. In: Proceedings of the 8th ACM Conference on Computer and Communications Security. pp. 245–254 (2001)
19. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 508–526. Springer, Heidelberg (Feb 2019). https://doi.org/10.1007/978-3-030-32101-7_30
20. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
21. PDecker, C., Russel, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin. See https://blockstream.com/eltoo.pdf (2017)
22. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. See https://lightning. network/lightning-network-paper. pdf (2016)
23. Richards, S., Wackerow, P.: Plasma (2021), https://ethereum.org/en/developers/docs/scaling/plasma/
24. Richards, S., Wackerow, P.: Sidechains (2021), https://ethereum.org/en/developers/docs/scaling/sidechains/

# A Appendix

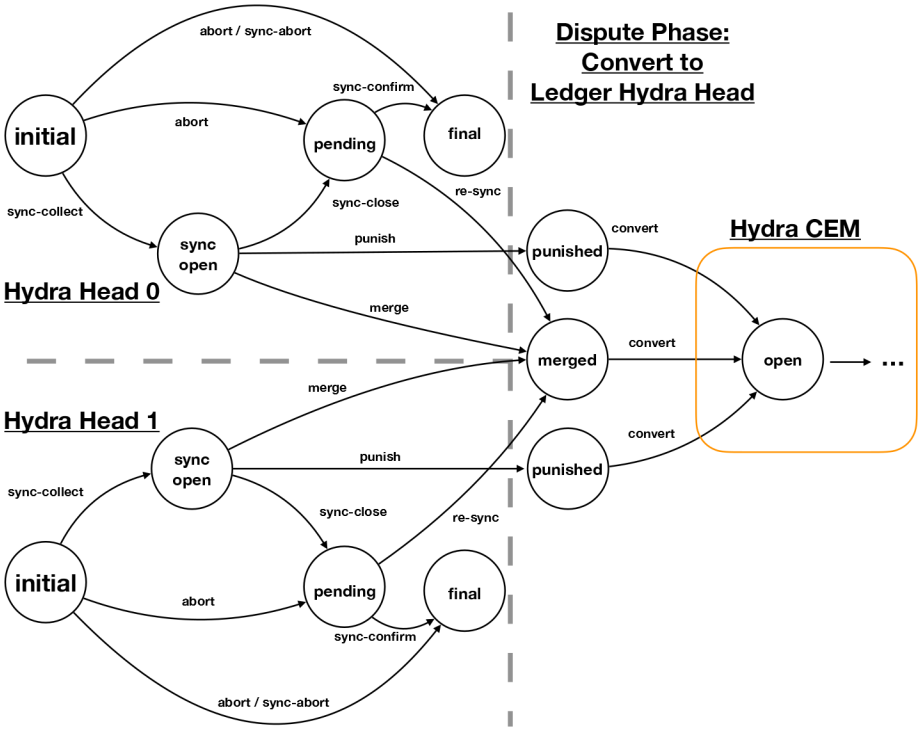


Fig. 15: Overview of all state transitions of the Interhead CEM. Note that most transitions are limited to one of the three phases of the construction.