State Machines across Isomorphic Layer 2 Ledgers

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

[1] Department of Mathematical and Computing Sciences,
School of Computing,
Tokyo Institute of Technology. {jourenko.m.ab@m, mario@c}.titech.ac.jp
[2] Input Output Global.
mario.larangeira@iohk.io
http://iohk.io

Abstract. With the ever greater adaptation of blockchain systems, smart contract based ecosystems have formed to provide financial services and other utility. This results in an ever increasing demand for transactions on blockchains, however, the amount of transactions per second on a given ledger is limited. Layer-2 systems attempt to improve scalability by taking transactions off-chain, with building blocks that are two party channels which are concatenated to form networks. Interaction between two parties requires (1) routing such a network, (2) interaction with and collateral from all intermediaries on the routed path and (3) interactions are often more limited compared to what can be done on the ledger. In contrast to that design, recent constructions such as Hydra Heads (FC'21) are both multi-party and isomorphic, allowing interactions to have the same expressiveness as on the ledger making it akin to a ledger located on Layer-2. The follow up Interhead Construction (MARBLE'22) further extends the protocol to connect Hydra Heads into networks by means of a "virtual" Hydra Head construction. This work puts forth an even greater generalization of the Interhead Protocol, allowing for interaction across different Layer-2 ledgers with a multitude of improvements. As concrete example, our design is modular and lightweight, which makes it viable for both full virtual ledger constructions as well as straightforward one-time interactions and payments systems.

Keywords: Blockchain, State Channel, Channel Network.

1 Introduction

Blockchain technology as introduced by Nakamoto [19] was a breakthrough in scaling byzantine consensus to a point where operation of decentralized ledgers among a large number of mutually distrustful parties became viable. While Bitcoin, Nakamoto's implementation of a decentralized ledger, remains one of the largest blockchain implementations by market capitalization [3] to date further

[3] https://coinmarketcap.com

blockchains such as Ethereum and Cardano expanded on the technology by enabling arbitrary smart contracts and state machines. This improved the utility of their ledgers which facilitated the creation of financial ecosystems. However, albeit blockchain's ability to scale to a seemingly arbitrary amount of users, the amount of transactions that can be performed on their ledgers is limited [5]. If there are more transactions being committed to a blockchain than its consensus mechanism can handle, transaction issuer can include a fee to their transactions to increase their priority. At times of high demand this can result in unfeasible high fees for an average transaction. One approach to mitigate this are Layer-2 protocols [21,20,6,4] such as Bitcoin's payment channel network Lightning [21]. Parties can move their coins into a Layer-2 structure which locks these coins on the ledger. Then, they can interact and perform payments with other parties that participate in the Layer-2 structure offchain, i.e. without requiring any transactions on the ledger itself. Only at the end, when a party wishes to move their coins back and unlock on the ledger another transaction is committed to the ledger that summarizes the transactions that occurred offchain. However, a common drawback of Layer-2 protocols is a lack of expressiveness of the interactions that can occur on Layer-2. For instance, payment channel networks are restricted to simple payments. State Channels [7,8] improve on that by allowing execution of smart contracts. Moreover, earlier versions of Layer-2 protocols operate on channels between two parties which can be concatenated by means of Hash Timelocked Contracts (HTLCs) [21] to perform payments or two parties in the network can connect by means of virtual channels [7,8,12,13], i.e. a channel that is created on Layer 2 instead of the ledger. This can be impractical since if two parties want to interact with another, it requires the intermediaries, i.e. the parties on the path between them, to lock away a large amount of collateral which ensures security of these protocols, however, such a path might not exist. Other approaches attempt to connect multiple parties, for instance Rollups – albeit not entirely Layer-2 as they require a small amount of data to be committed to the ledger per transaction – can directly connect an arbitrary amount of parties, however, making the expressiveness of interactions on Rollups on par to the ledger is ongoing research. Hydra [4] is a Layer-2 protocol that forms an isomorphic state channel called Hydra *Head* for an arbitrary amount of parties which allows interaction to have the same expressiveness as on the ledger itself. This makes Hydra Heads akin to a ledger located in Layer-2. However, while interaction between different Hydra heads by utilizing intermediaries is possible it is either limited to payments (HTLCs) or requires iterative construction of virtual Hydra heads [14], which construction is heavy as it requires partial execution of the Hydra Head state machine. Moreover, the construction is complex making it difficult to verify its security and also making it prone to implementation errors. It is inflexible because all UTxO that are available on the Interhead have to be moved into it at the very beginning of the protocol. Since it is tightly related to the Hydra State Machine construction, adaptation of any changes to the Hydra State Machine would require additional work and careful consideration to ensure security of the Interhead construction.

Our Contributions. The aim of this work is to create a lightweight *ad-hoc ledger* to enable arbitrary interactions between parties on separate Layer-2 ledgers, i.e. Layer-2 structures containing an arbitrary amount of parties and which have the same expressiveness as a smart-contract capable ledger. Our work is based on the Interhead [14] construction and in fact is a generalized version of it. Similarly we assume two Layer-2 ledger based on the Unspent Transaction Output (UTxO) paradigm and utilize a set of intermediaries, i.e. parties that participate in both Layer-2 ledgers, to facilitate payments as well as execution of arbitrary state machines. However, in addition to the previous work our construction provides a multitude of improvements: (1) There is no time limit to the ad-hoc ledger, (2) setup is done only once and can be reused for future interactions, (3) UTxO can be moved between Layer-2 ledgers and the ad-hoc ledger at any time compared to only at the beginning and the end of the Interhead construction making the ad-hoc ledger more flexible, (4) disputes are local only affecting individual UTxO instead of the whole structure, (5) a modular and therefore significantly simpler construction. (6) While we present the core of our construction in this work, we also present multiple potential extensions to further improve on the scalability of the construction. Additionally, as with the Interhead construction, collateral does not need to be paid by single individual intermediaries but instead any collateral can be paid by multiple intermediaries. Although a tradeoff of our construction is that we require interaction with all intermediaries for each transaction on the ad-hoc ledger, we are able to execute the Hydra Head state machine within it creating a virtual Hydra Head where interaction with the intermediaries is no longer necessary. This gives us the same function and benefits as the Interhead allowing our construction to be both a generalization of the Interhead construction as well as the Hydra Head construction. While our work assumes a UTxO based ledger we argue that any Layer-2 ledger that can implement an adaptation of the state machine presented in this work can execute our construction thus it is not limited to be used with Hydra Heads alone, but aims to enhance interoperability between any Layer-2 ledgers.

Related Work. Layer 2 or *offchain* structures are scalability solutions for ledgers. Early approaches are payment channels [20,6] where two parties, first, lock coins on the ledger via a transaction and then perform an offchain protocol to perform payments between another without requiring to commit any further transactions. Only at the very end, one last transaction is committed to the ledger that summarizes all payments and unlocks the two parties' coins. Protocols such as Hash Timelocked Contracts (HTLCs) [21] enable payments across multiple adjacent channels allowing for the formation of payment channel networks. An efficiency requirement for Layer 2 structures and protocols is that when performing $\mathcal{O}(n), n \in \mathbb{N}$ transactions then only $\mathcal{O}(1)$ transactions are committed to the ledger. More recent approaches such as Hydra [4] allow for an arbitrary amount of parties to interact offchain with the same expressiveness as on the ledger itself instead of being limited to simple payments, effectively forming a sub-ledger on

Layer 2. Another notable approach are Rollups [4] where an arbitrary amount of parties can interact offchain with a few caveats: To our knowledge, rollups based on Zero-Knowledge proofs do not yet support full expressiveness of the ledger although there is active research to achieve this. Moreover, for reasons of data availability, each transaction within a rollup produces some data that has to be committed to the ledger therefore it is akin to a Hybrid protocol rather than a full Layer 2 protocol. The Interhead [14] allows parties across two Hydra Heads to interact with another with the aid of intermediaries, i.e. parties participating on both Hydra Heads, by creation of a virtual Hydra head. Our work aims to provide a lightweight, flexible and modular generalization to the Interhead construction not only allowing for the creation of a virtual Hydra Head, but also providing a low-overhead framework for brief interactions.

2 Background

Notation. In this work we consider structured data. If we assume a value $\beta \in \mathcal{B}$ of form $(\beta_0, \ldots, \beta_n)$, $n \in \mathbb{N}$, then $\beta.\beta_i$ is the value of β with label β_i, $i \in \mathbb{N}$, $0 \leq i \leq n$. Moreover, parties within a protocol are denoted using $\mathcal{P}$. Lastly $\mathcal{H}$ denotes a cryptographic hash function.

Signatures. We assume a cryptographic signature scheme [9,10,1] with existential unforgeability under a chosen message attack (EU-CMA) consisting of algorithms ($\mathsf{key\_gen}$, verify, sign). Then $\mathsf{key\_gen}(1^\lambda) = (vk, sk)$ generates a verification key vk and a private key sk using security parameter 1^λ, $\mathsf{sign}(sk, m) = \sigma$ takes sk and a message $m \in \{0,1\}*$ as input and creates a signature $\sigma \in \{0,1\}*$ and $\mathsf{verify}(vk, m', \sigma')$ takes vk, a message m and a signature σ' as input and outputs 1 on successful verification and 0 otherwise. We assume a secure multisignature scheme [11,18] with algorithms ($\mathsf{ms\_setup}$, $\mathsf{ms\_key\_gen}$, $\mathsf{ms\_agg\_vk}$, $\mathsf{ms\_sign}$, $\mathsf{ms\_agg\_sign}$, $\mathsf{ms\_verify}$). Algorithm $\mathsf{ms\_setup}(1^{\lambda'}) = \Pi$ creates public parameters Π, algorithm $\mathsf{ms\_key\_gen}(\Pi) = (vk, sk)$ creates a new set of verification key vk' and private key sk, algorithm $\mathsf{ms\_agg\_vk}(\Pi, V) = \mathsf{vk}_{agg}$ takes Π and a set of verification keys V as input and outputs an aggregate verification key vk_{agg}, algorithm $\mathsf{ms\_sign}(\Pi, sk, m) = \sigma$ creates a signature σ on message m, algorithm $\mathsf{ms\_agg\_sign}(\Pi, V, S, m) = \sigma_{agg}$ aggregates a set of signatures S on message m to an aggregate signature σ_{agg} and lastly $\mathsf{ms\_verify}(\Pi, m', vk_{agg}, \sigma_{agg})$ verifies an aggregate signature on a message to a aggregate verification key where it outputs 1 upon success and 0 otherwise.

The UTxO Ledger and Extensions. In UTxO based ledgers such as Bitcoin [19] coins that are in circulation are represented using a tuple (b, ν) where $b \in \mathbb{N}$ is an amount of coins and ν is a verification script that evaluates to a value in $\{0,1\}$ such that the coins within a UTxO can be spent if presented a witness w where $\nu(w) = 1$. The ledger's state is represented as a set U of all currently circulating UTxO. Transactions can be used to spend UTxO and thereby perform a state

[4] https://ethereum.org/en/developers/docs/scaling/zk-rollups/

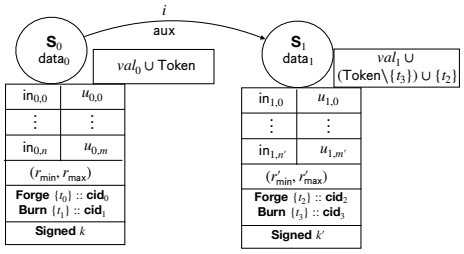


Fig. 1: A general state machine transition moving it from state S_0 to S_1 on input symbol i and auxiliary input data aux. Each state is represented by a UTxO on the ledger, in the case of S_0 with data field data_0, coins val_0 and (non-) fungible token Token. The box below a state represents constraints to transactions creating that UTxO. The left-hand side contains UTxO inputs that are spent and the right-hand side UTxO outputs that are created. The transaction is valid only in time $(r_{\mathsf{min}}, r_{\mathsf{max}})$, burns token t_1 while minting token t_0 and is signed corresponding to public verification key k.

transition on the ledger. A transaction is of form $(\mathsf{In}, \mathsf{Out}, t)$ where In is a set of tuples of form (ref, w) where ref is a pointer to an UTxO that exists on the ledger and w is a witness as above, Out is a list of new UTxO and timelock $t \in \mathbb{N}$ is a point in time such that a transaction can be applied on the ledger only after time t. A transaction is included in the ledger after being submitted after at most time $\Delta \in \mathbb{N}$. Note that even though UTxO might be similar, the ledger ensures that all UTxO are unique by assigning them unique addresses. The Extended UTxO model [2] adds an arbitrary data field $\delta \in \{0,1\}^*$ to UTxOs. Moreover, the verification script ν is extended to additionally receive δ as well as a context $\mathsf{ctx} \in \{0,1\}^*$ consisting of the transaction that creates the UTxO as well as the UTxO that are referenced within the transaction's inputs. Doing so ν can enforce constraints on transactions. Lastly, timelocks are extended to form time ranges $[r_0, r_1]$, $r_0, r_1 \in \mathbb{N}$ where a transaction can be committed onto the ledger within this time range. It has been shown [2] that there exists a weak bi-simulation between Constraint Emitting Machines (CEM), which are state machines derived from Mealy automata, such that it is possible to execute state machines defined as CEMs as in Figure 1 on EUTxO based ledgers. Further work [3] adds multi-asset support such that they not only contain coins, but also fungible and non-fungible token. In this work we consider EUTxO with multi-asset support, but for simplicity refer to them as UTxO.

3 Overview

We assume two layer 2 ledgers $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ created on a common ledger $\mathcal{L}$. Let parties $P_b = \{\mathcal{P}_{b,0}, \ldots, \mathcal{P}_{b,i}\} \ldots, \mathcal{P}_{b,n_b}$ for $b \in \{0,1\}$ be an arbitrary, non-empty

subset of the parties who participate in ledgers $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ respectively and $P = P_0 \cup P_1$, where $|P| = n$, $|\mathcal{P}_0| = n_0$ and $|\mathcal{P}_1| = n_1$ i, n, n_0, $n_1 \in \mathbb{N}$. Let P_{int} $= P_0 \cap P_1 \neq \varnothing$ with $|P_{\mathsf{int}}| = n_{\mathsf{int}}$, $n_{\mathsf{int}} \in \mathbb{N}$ be the set of intermediaries.

Communication Model and Time. We assume synchronized communication between parties which happens in rounds such that, if a message is send within one round it is available to the recipient at the beginning of the next round. We assume a relation between communication rounds and time [15,16,17].

Adversarial Model. We assume an malicious adversary $\mathcal{A}$ who can statically corrupt $n-1$ out of n parties where $n \in \mathbb{N}$. Corrupted parties leak their internal state including their secret keys to the adversary and communication from and $\mathcal{A}$ receives all communication from and to that party. $\mathcal{A}$ can dictate the corrupted party's behaviour and make them deviate from the protocol arbitrarily.

Layer 2 Ledgers. We assume the existence of a Layer 2 ledger construction for a UTxO based ledger. Moreover, we assume that the Layer 2 ledger can implement CEMs or allows for execution of state machines with sufficient expressiveness. As is the case with regular ledgers, we assume that all UTxO on the Layer 2 ledger are unique. Moreover, if a Layer 2 ledger $\mathcal{L}^2$ is instantiated on a UTxO based ledger $\mathcal{L}$, then there exists $\Delta_{\mathcal{L}^2} \in \mathbb{N}$ such that any UTxO can be moved from $\mathcal{L}^2$ to $\mathcal{L}$ within time $t \in \mathbb{N}$ with $t \leq \Delta_{\mathcal{L}^2}$. We note that a construction that fulfills these requirements is Hydra [4].

Semantic UTxO Equality. Any well defined ledger makes sure that each UTxO is unique by assigning unique addresses to each newly created UTxO. In our construction we are considering multiple instances of the same UTxO where two UTxO are *semantically equal* if they are equal except for their address. For instance, we consider two UTxO that each award 5 coins to a party $\mathcal{P}$ to be semantically equal even though their addresses are distinct.

3.1 The Goal

The aim of this work is to allow interaction between an arbitrary set of parties P with the same expressiveness as on any Layer 2 ledger. In the following we denote interaction between parties in P as interaction on an *ad-hoc* ledger $\mathcal{L}_P$. We define the properties we desire in our construction consistent with related work as follows.

Definition 1 (Offchain Efficiency). *No transactions are committed to $\mathcal{L}$ except in the case of dispute where $\mathcal{O}(1)$ transactions are committed to $\mathcal{L}$.*

Definition 2 (Liveness). *There exists $t \in \mathbb{N}$ such that upon a party's request any UTxO in $\mathcal{L}_P$ can be made available on $\mathcal{L}$ or $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ after at most time t. If there exists a honest intermediary the same holds true for collateral.*

Definition 3 (Balance Security). *Any honest party loses access to their collateral and UTxO without their consent at most with negligible probability.*

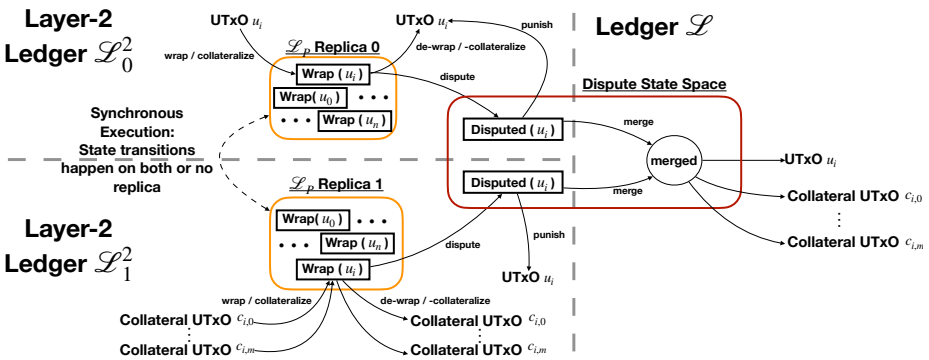


Fig. 2: Overview of our Setting with two Layer-2 ledger $\mathcal{L}_0^2$, $\mathcal{L}_0^2$ and the ledger $\mathcal{L}$ they are created on all separated using dashed lines. We display the lifecycle of a UTxO u_i which is moved into replicas of the ad-hoc ledger $\mathcal{L}_P$ and which can either be moved out of it regularly or through dispute.

Consensus. If a UTxO can be spent by two different transactions, then these transactions are in conflict as any UTxO can only be spent once. In the remainder we assume a mechanism that decides which transactions to perform, e.g. a leader-based approach as in approach in Hydra [4]. If we relax security by allowing the adversary to corrupt only less than a third of participants, then another approach are byzantine consensus protocols as HotStuff [22]. However, this is an orthogonal problem to our work and we do not further address it in this work.

3.2 Approach

The construction consists of three components. (1) UTxOs are moved into and out of $\mathcal{L}_P$. (2) Perform arbitrary transactions that consume and create UTxOs in $\mathcal{L}_P$. (3) Any UTxO in $\mathcal{L}_P$ can be disputed and made available on the underlying ledger $\mathcal{L}$ or in $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$. Recall that in the EUTxO model, coins, (non-) fungible token as well as CEMs, i.e. state machines, are represented as UTxO such that showing the above steps for any UTxO is sufficient to show that it is not only possible to perform payments but also to execute CEMs on $\mathcal{L}_P$.

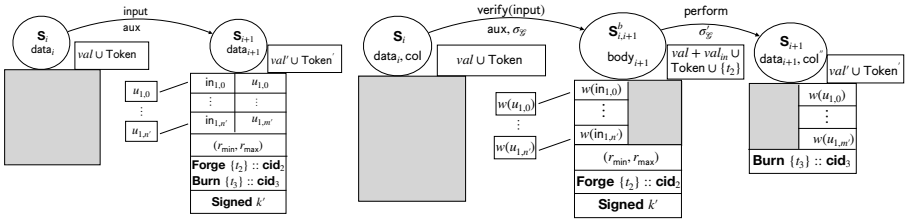
Wrapped UTxO. Figure 2 illustrates the lifecycle of any UTxO in $\mathcal{L}_P$. Each ledger $\mathcal{L}_b$ maintains a *replica* $\mathcal{R}_b$ – a copy – of $\mathcal{L}_P$. In the following we look at the example of moving a UTxO from $\mathcal{L}_0^2$ to $\mathcal{L}_P$. We consider a UTxO to be in $\mathcal{L}_P$ by *wrapping* it inside a CEM that (1) makes sure that if a UTxO is moved into $\mathcal{L}_P$ on legder $\mathcal{L}_i^2$, $i \in \{0,1\}$, then it is moved into $\mathcal{L}_P$ on $\mathcal{L}_{1-i}^2$ by collecting collateral from a subset of the intermediaries $P_i \subseteq P_{\mathsf{int}}$ on $\mathcal{L}_{1-i}^2$. (2) Likewise it can be *de-wrapped* and moved into $\mathcal{L}_j^2$, $j \in \{0,1\}$ by returning the collateral to the intermediaries on $\mathcal{L}_{1-j}^2$.

Dispute Mechanism. Correctness is facilitated through collaboration with the intermediaries in P_{int}. If any intermediary in P_{int} misbehaves or fails to collaborate,

a UTxO can be *disputed* by any participating party. A dispute has two outcomes: (1) If there is at least one honest intermediary they move both instance of the wrapped UTxO from $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ respectively and onto the underlying ledger $\mathcal{L}$. Afterwards, they can use both UTxO as input into a *merge* transaction which has the original UTxO in its outputs as well as all collateral that was committed with it. (2) If no intermediary is honest such that none perform the steps in (1), then after a timeout enforced through a timelock, two semantically equal instances of the UTxO are moved into both $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ using the collateral of the intermediaries to finance it and in the process punish the intermediaries. This ensures that the UTxO is always available to their owners independent on which Layer 2 ledger they participate in.

Atomic Transactions. Intermediaries collaborate to perform transactions on $\mathcal{L}_P$ atomically, meaning it is performed on both or on no replica. Otherwise, if a UTxO is spent on $\mathcal{R}_0$ but not on $\mathcal{R}_1$, it can be spent by a different transaction on $\mathcal{R}_1$ effectively double spending the UTxO in which process the collateral of the intermediaries is implicitly consumed and lost. Atomic transactions have to be performed for spending already wrapped UTxO, as well as for wrapping and de-wrapping of UTxO. Transactions are performed atomically by splitting them into two steps, where each step is performed by a dedicated transaction. (1) First, we verify that the transactions can be performed through the *verify-transaction*. The verify-transaction collects all the (wrapped) UTxOs on both replica as well as any witnesses, and forges token if necessary. Moreover it evaluates the UTxOs verification scripts. Note that this step is reversible, i.e. we can create another transaction that re-creates all input UTxOs by creating semantically equal ones within its outputs and burns all forged tokens. This transaction requires an aggregate signature signed by all intermediaries. We proceed only if this transaction has been performed on both replica. (2) Only afterwards we can be sure that the transaction can be done on both replica ensuring it is atomic. This is done through a perform transaction that creates all UTxO within its outputs and burns any token if required. To ensure that any honest intermediary can prevent wrongful execution of the perform transaction, i.e. before the replica are synchronized, we require an aggregate signature signed by all intermediaries to create the perform transaction. This aggregate signature is the only witness required to perform the transactions. This step is irreversible, we might not be able to create a transaction that can reforge burned token, or claim all UTxO we created as input as they might be spent by different transactions by then.

We can resolve disputes through a merge-transaction, (1) either if the UTxO in both replica are in the same state, (2) one replica has performed only the verify-transaction while the other hasn't since we can revert this step by outputting the UTxO in the merge transaction's outputs and (3) if the perform transaction was performed on one replica while only the verify-transaction was performed on the other replica since we have the required witness, i.e. the aggregate signature, to do the perform transaction bringing the UTxO of both replica into the same state. Acting as mentioned above any honest intermediary can ensure that a merge-transaction can be performed on disputed UTxO.



(a) The original transaction. (b) Atomic transaction on each replica.

Fig. 3: Derivation of a atomic two-step transaction in Figure 3b from an arbitrary transaction in Figure 3a.

Limitations. We require UTxO that are moved to $\mathcal{L}_P$ to be collateralized, i.e. the intermediaries have to commit collateral equal to the number of coins and token present. This limits us to UTxO that contain coins or fungible token, however, we cannot move non-fungible token into $\mathcal{L}_P$ without any additional workaround.

4 The Ad-Hoc Ledger State Machine

In the following we give a description of the state machine that governs the lifecycle of each UTxO within an ad-hoc ledger $\mathcal{L}_P$ between parties P. Note that while we do not specify the exact storage of data, however, to reduce the size of UTxO a potential data structure are Patricia Merkle trees[5].

4.1 Setup

Recall that we aim to setup an ad-hoc ledger between parties P with the help of intermediaries P_{int}. For setup, each party $\mathcal{P}$ creates an individual key pair $(vk_{\mathcal{P}}, sk_{\mathcal{P}})$. Moreover, the parties collaborate to setup public parameter Π_P and use them to create aggregate verification key V_P where $(\mathsf{sk}_{P,\mathcal{P}}, \mathsf{vk}_{P,\mathcal{P}})$ are the individual keys of $\mathcal{P}$. Analogously the intermediaries collaborate to setup public parameter Π_{int} to create aggregate verification key V_{int} where $(\mathsf{sk}_{\mathsf{int},\mathcal{P}'}, \mathsf{vk}_{\mathsf{int},\mathcal{P}'})$ are the individual keys of $\mathcal{P}' \in P_{\mathsf{int}}$. Then, the parties sample a random nonce $r \in \mathbb{N}$ and negotiate a dispute time $t_d \geq \Delta_{\mathcal{L}^2} + \Delta$. This data is stored within the data field of each wrapped UTxO. A cryptographic hash h_{MP} of the execution parameters, public keys and nonce r serves to bind them to the ad-hoc ledger and in addition they are used as a unique identifier for the ad-hoc ledger itself.

4.2 Atomic Transactions

Transactions are executed on all replica atomically, i.e. they are executed either on all or none, by splitting them up into two transactions, (1) the verify-transaction verifying that the transaction can be executed on each replica and

[5] https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/

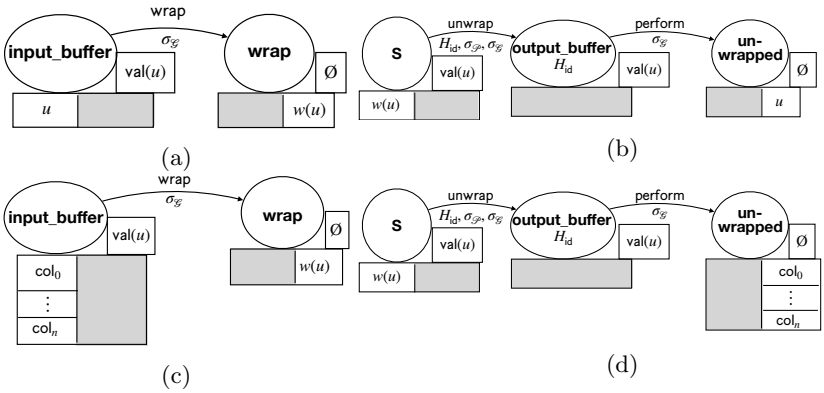


Fig. 4: Illustration of wrapping a UTxO in figures 4a and 4c and unwrapping in figures 4b and 4d.

(2) the perform-transaction which executes the transaction. This is illustrated in Figure 3b where the original-transaction depicted in Figure 3a is executed atomically. These transactions are executed on all replica to keep their states equal. Both transactions require an aggregate signature corresponding to verification key V_{int}. (1) The first transaction is labeled verify(input) where input is the label of the original-transaction that should be performed. The transaction collects all UTxO that are required for the transaction in its inputs while verifying their validator scripts and forges all token that are required. It verifies the validator script of the input transition using auxilliary information aux. However, note that since the verify-transaction does not contain the outputs of the original-transaction, we cannot directly verify the transaction constraints that are required by the validator. Instead, we store the body of the original-transaction that we perform in the body_{i+1} verify that it confirms the constraints and that the inputs of the verify-transaction confirm with the original-transaction. (2) After the verify-transaction is performed on each replica we proceed with the perform-transaction. This transaction burns all required token and creates the UTxO outputs of the original-transaction. We use body_{i+1} that is available through $S^b_{i,i+1}$'s data field to verify that the perform-transaction is consistent with the original-transaction. The field col stores how the coins and tokens associated with that UTxO $val \cup \mathsf{Token}$ are collateralized by the intermediaries as discussed in Section 4.4.

4.3 Wrapping UTxO

All UTxO that are used within $\mathcal{L}_P$ are wrapped using a state machine that has two purposes. (1) It manages the lifecycle of the UTxO as shown in Figure 2 and ensures it is only spent through verify-/ and perform-transactions and (2) it ensures sufficient collateral was committed as well as it tracks how much collateral was committed by the individual parties. Figures 4a and 4c depicts how UTxO are wrapped and made available on $\mathcal{L}_P$ wheres figures 4b and 4d depict

how they are unwrapped and moved out of $\mathcal{L}_P$ back into either $\mathcal{L}_0^2$ or $\mathcal{L}_1^2$. All operations have to be done atomically and thus are executed using the framework described in Section 4.2. When wrapping a UTxO u, it is committed into the input-buffer transaction on the ledger it originates from, whereas for the other replica the intermediaries $\mathcal{P}_{\mathsf{int},0}$, ..., $\mathcal{P}_{\mathsf{int},n_{\mathsf{int}}}$ commit collateral col_0, ..., $\mathsf{col}_{n_{\mathsf{int}}}$ respectively such that $\mathsf{val}(u) \leq \sum_{i=0}^{n_{\mathsf{int}}} \mathsf{col}_i$ where $\mathsf{val}(u)$ is the amount of coins and token in u. Only after the input_buffer transaction has been committed to both replica, the intermediaries collaborate to create the wrap-transactions which are analogous to the perform-transaction in Section 3b and make the wrapped UTxO available in $\mathcal{L}_P$. Lastly, each wrapped UTxO contains a nonce u_{id} that is unique to $\mathcal{L}_P$ to make it uniquely identifiable, however, which is equal for the same wrapped UTxO on each replica. Unwrapping of a UTxO is analogous with a few differences. For one, we require that the unwrap-transaction in addition contains information H_{id} which designates that the UTxO will be moved to $\mathcal{L}_{H_{\mathsf{id}}}^2$ as depicted in Figure 4b and the collateral associated with it is released on the other layer 2 ledger as depicted in Figure 4d. Moreover, to trigger unwrapping of a UTxO we additionally require a group signature corresponding to the aggregate verification key of all participants V_P of the message (unwrap, u_{id}, H_{id}).

4.4 Collateral

The wrapping itself is a unique UTxO living in $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$. It's datafield contains the amount of collateral committed by each intermediary. Let $\mathsf{col}(\mathcal{P}, w(u))$ be the collateral intermediary $\mathcal{P}$ has contributed to wrapped UTxO $w(u)$. If a transaction consumes wrapped UTxOs $w_{\mathsf{in}}(u_0)$, ..., $w_{\mathsf{in}}(u_n)$, and creates wrapped UTxOs $w_{\mathsf{out}}(u_0)$, ..., $w_{\mathsf{out}}(u_m)$ $n, m \in \mathbb{N}$, then for each intermediary $\mathcal{P}_i$, $0 \leq i \leq n_{\mathsf{int}}$ holds that the sum of their committed collateral does not change, i.e. $\sum_{j=0}^{n} \mathsf{col}(w_{\mathsf{in}}(u_j)) = \sum_{l=0}^{m} \mathsf{col}(w_{\mathsf{in}}(u_l))$. It has to hold that each wrapped UTxO remains sufficiently collateralized, i.e. the inequation $\mathsf{val}(u) = \sum_{i=0}^{n_{\mathsf{int}}} \mathsf{col}_i$ holds. How collateral is distributed within the new UTxOs is to be negotiated between the intermediaries. In Layer-2 protocols intermediaries receive a fee for locking their collateral. While not addressing it in detail we argue that we can adapt the handling of fees from the Interhead [14] where intermediaries receive fee proportional to the collateral locked whereas parties pay out a fee to the intermediaries proportional to the value of UTxO they request moving to $\mathcal{L}_P$.

4.5 Dispute

Dispute resolution is similar to Interhead Hydra [14]. At any point, any party can create a transaction that consumes a wrapped UTxO and outputs a semantically equal UTxO to which a dispute flag is added. A dispute might be required if the intermediaries fail to perform a transaction atomically on all replica, or if the intermediaries stop collaborating to perform any further transaction which is required to ensure liveness.. A UTxO with such a flag can be spent in two ways. (1) As depicted in Figure 5a a merge-transaction consumes one instance of the

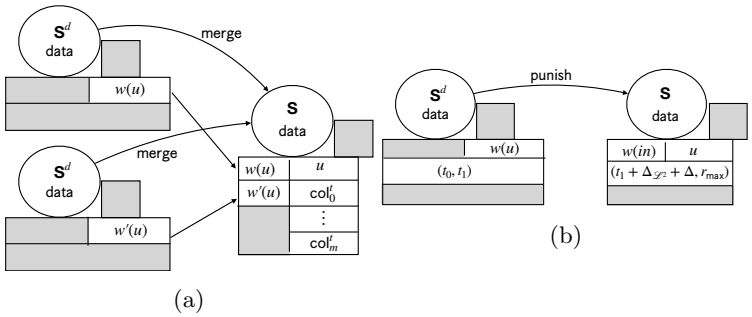


Fig. 5: Figure 5a shows dispute resolution through merge of its two wrapped UTxO instances. Figure 5b shows timeout and punishment of the intermediaries.

wrapped UTxO from each replica and outputs the UTxO as well as all collateral associated with it. This requires moving the flagged UTxO out of $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ and to the underlying ledger which can be done by any party which takes at most time $\Delta_{\mathcal{L}^2}$. Recall that committing any transaction on the ledger takes time Δ. Therefore any intermediary can perform this within time $\Delta_{\mathcal{L}^2} + \Delta$. However, the above dispute resolution requires that both wrapped UTxO are available. If dispute is triggered while a UTxO is involved in a atomic transaction it might be only available on one replica. Recall the atomic transaction in Figure 3b where a UTxO is first in state S_i, then a verify-transaction moves it to a buffer state $S^b_{i,i+1}$ and lastly the UTxO is moved with a perform-transaction to state S_{i+1}. Since the intermediaries synchronize after each transaction as mentioned in Section 4.2 and described in Section 5 the states can only diverge by one state transition and we can proceed as follows. If both states are at least in state $S^b_{i,i+1}$ we already verified on all replica that the original-transaction can be performed thus the merge-transaction outputs a UTxO in state S_i and burning token as well as outputting wrapped UTxO on the ledger consistent to body_{i+1}. Otherwise one UTxO is in state S_i while the other is in state $S^b_{i,i+1}$. Then the merge transaction outputs a UTxO in state S_i while reversing the verify-transaction, i.e. it for each UTxO in the verify-transaction's inputs it outputs semantically equal wrapped UTxO on the ledger, as well as it burns all token that were forged in that transaction. (2) As shown in Figure 5b a timelock expires after time $\Delta_{\mathcal{L}^2} + \Delta$ which allows the flagged UTxO to be spent by a transaction that consumes it and outputs the UTxO directly and without outputting any collateral. Note that if a UTxO is in state $S^b_{i,i+1}$ the UTxO that will be output is in state S_{i+1}.

4.6 Extensions

Batch Transaction. To improve efficiency of the construction multiple wrap and de-wrap transactions could be batched into one transaction. Moreover, instead of requiring wrapped UTxO as input, a sync-transaction can take UTxO and collateral as input and implicitly wrapping the UTxO in the same step.

Multiple Replica. Our construction can be extended to $n \geq 2$, $n \in \mathbb{N}$ replica. As in the case of $n = 2$ all replica require to synchronize at *sync-transactions* and merge-transactions have to be adapted to merge not two but n disputed UTxOs. However, this naive extension to multiple replica requires that the collateral committed by each intermediary for each UTxO in each replica is equal. The option of having variable collateral is left as an open question.

Recovery from Disputes. If UTxOs are disputed within each replica, but are either in the same state or can be brought into the same state through reversing a sync-transaction or doing a perform transaction they could be moved back into their respective replicas and thus into L_P thus resolving the disupte. However, to ensure correctness and liveness, we require an aggregate signature of the intermediaries as well an aggregate signature of all remaining parties, i.e. all parties must verify and consent to recover from a dispute as otherwise corrupted parties might prevent a correct dispute resolution.

Virtual Ledgers. Since our construction requires interaction with all intermediaries in P for each transaction it cannot be considered a virtual ledger. However, layer 2 ledger constructions such as Hydra [4] can be performed within our framework effectively creating virtual ledgers.

5 The Protocols

We require that all transactions performed on $\mathcal{L}_P$ are either performed on all replica or on none which is ensured through ATOMIC_TRANSACTION protocol shown in Algorithm 1 which is executed by the intermediaries. This protocol is executed for general transactions and (un-) wrapping of UTxOs. If this fails, there would be UTxO that are on only one replica, i.e. UTxO are not spent on the replica that did not perform the transaction, whereas new UTxO are created only on the replica that did perform the transaction. Any UTxO that exists on only one replica can be claimed by their owner by setting a dispute flag on the UTxO which allows it to be claimed directly after time $\Delta_{\mathcal{L}^2} + \Delta$. A dispute might also be triggered to ensure liveness, if the intermediaries stop collaboration to perform further transactions. However, if a disputed UTxO exists on all replica, the intermediaries can perform the DISPUTE_UTXO protocol shown in Algorithm 2 to move the UTxO out of $\mathcal{L}_P$ and into the common ledger $\mathcal{L}$.

Atomic Transactions. Algorithm 1 describes how all Intermediaries collaborate to perform a transaction atomically on all replica. Whenever a participant of $\mathcal{L}_P$ requests a transaction to be performed they submit the original transaction tr_o. First, we verify whether tr_o is a valid transaction in lines 2 and 3 tr_o and terminate if this is not the case. In line 4 we derive the $\mathsf{verify-transactions}$ $\mathsf{tr}_{\mathsf{v},0}$ and $\mathsf{tr}_{\mathsf{v},1}$ for the replicas in $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ respectively and in line 5 we derive the $\mathsf{perform-transactions}$ $\mathsf{tr}_{\mathsf{v},0}$ and $\mathsf{tr}_{\mathsf{v},1}$. In lines 6 - 8 the intermediaries collaborate to create group signatures for the $\mathsf{verify-transactions}$, commit them

Algorithm 1 Transaction Protocol

1: **function** ATOMIC_TRANSACTION(tr_o)
2: **if** $\neg$VERIFY(tr_o) **then return**
3: **end if**
4: $(\mathsf{tr}_{\mathsf{v},0}, \mathsf{tr}_{\mathsf{v},1}) \leftarrow$ VRFY_TR(tr_o)
5: $(\mathsf{tr}_{\mathsf{p},0}, \mathsf{tr}_{\mathsf{p},1}) \leftarrow$ PRFRM_TR(tr_o)
6: AGGREGATE_SIG($V_{\mathsf{int}}, \mathsf{tr}_{\mathsf{v},0}, \mathsf{tr}_{\mathsf{v},1}$)
7: COMMIT($(\mathcal{L}_0^2, \mathsf{tr}_{\mathsf{v},0}), (\mathcal{L}_1^2, \mathsf{tr}_{\mathsf{v},1})$)
8: WAIT_COMITTED($\mathsf{tr}_{\mathsf{v},0}, \mathsf{tr}_{\mathsf{v},1}$)
9: AGGREGATE_SIG($V_{\mathsf{int}}, \mathsf{tr}_{\mathsf{p},0}, \mathsf{tr}_{\mathsf{p},1}$)
10: COMMIT($(\mathcal{L}_0^2, \mathsf{tr}_{\mathsf{p},0}), (\mathcal{L}_1^2, \mathsf{tr}_{\mathsf{p},1})$)
11: WAIT_COMITTED($\mathsf{tr}_{\mathsf{p},0}, \mathsf{tr}_{\mathsf{p},1}$)
12: **end function**

Algorithm 2 Dispute Protocol

1: **function** DISPUTE_UTXO(u)
2: **if** $\neg$DISPUTED(u) **then return**
3: **end if**
4: DECOMMIT($\mathcal{L}_0^2, u$)
5: DECOMMIT($\mathcal{L}_1^2, u$)
6: WAIT_DECOMMITTED(u)
7: $\mathsf{tr}_{\mathsf{m}} \leftarrow$ MERGE_TX(u)
8: COMMIT($\mathcal{L}, \mathsf{tr}_{\mathsf{m}}$)
9: **end function**

Fig. 6: Algorithm 1 is executed by intermediaries to perform transactions whereas Algorithm 2 is done by any one intermediary to resolve a dispute.

and wait until they are confirmed by the respective layer 2 ledgers. In line 6, AGGREGATE_SIG takes an aggregate verification key and two transactions as input and outputs aggregate signatures for both transactions corresponding to that verification key. Then, COMMIT takes tuples of form $(\mathcal{L}, \mathsf{tr})$ as input and commits transaction tr onto (layer 2) ledger $\mathcal{L}$. In line 8 WAIT_COMMITTED takes a list of transactions as input and makes the protocol participants wait until these transactions are processed on their respective ledgers. After this is done the same is repeated for the $\mathsf{perform} - \mathsf{transactions}$ in lines 9 - 11.

Dispute. Algorithm 2 describes how a dispute can be resolved by any one intermediary without them losing their collateral. This algorithm can only be executed if a disputed UTxO is present on all replica. The algorithm takes a wrapped UTxO as input. First, the intermediary checks whether the UTxO's dispute flag is set in lines 2 - 4 and terminates the algorithm otherwise. If the UTxO is disputed the wrapped UTxOs are decommitted from $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ respectively and moved to the common ledger $\mathcal{L}$ within time $\Delta_{\mathcal{L}^2}$. Then, in line 6 the intermediary observes $\mathcal{L}$ and waits until both replica of the UTxO are present on $\mathcal{L}$ after which in line 7 a $\mathsf{merge} - \mathsf{transaction}$ for the disputed UTxO is created that takes the decommitted wrapped UTxOs and outputs the UTxO as well as all collateral that is associated with it as depicted in Figure 5a. Lastly in line 8 the $\mathsf{merge} - \mathsf{transaction}$ is committed to $\mathcal{L}$ and is processed within time Δ.

6 Analysis

Theorem 1 (Offchain Efficiency). *If $\mathcal{L}^2$ can de-commit a UTxO to the ledger in $\mathcal{O}(1)$ transactions, $\mathcal{L}_P$ has offchain efficiency.*

Proof. The only occasion in which UTxO are committed to $\mathcal{L}$ is when an intermediary executes DISPUTE_UTXO in Algorithm 2. In that case the disputed UTxO is de-commited from both, $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ which happens in $\mathcal{O}(1)$ transactions. Afterwards one merge-transaction is committed to the ledger.

Theorem 2 (Liveness). *$\mathcal{L}_P$ has the liveness property.*

Proof. At any point within a UTxOs lifecycle within $\mathcal{L}_P$ including during wrapping and unwrapping it can be disputed by any party including any intermediary. If there is a honest intermediary, they will proceed to make the UTxO and the associated collateral available on the ledger by executing Algorithm 2. This happens within time $\Delta_{\mathcal{L}^2} + \Delta$. Otherwise, at time $\Delta_{\mathcal{L}^2} + \Delta$ a semantically equal UTxO is available in Layer 2 ledgers $\mathcal{L}_0$ and $\mathcal{L}_1$ respectively.

Theorem 3 (Balance Security). *$\mathcal{L}_P$ has the balance security property.*

Proof. In the following we assume that all UTxO are well formatted, i.e. a owner of a UTxO has given consent to spent the UTxO to any computationally polynomially bound party that can compute a valid witness for the UTxO. Moreover, in the following we consider a honest party $\mathcal{P}$ and a UTxO u that is present in $\mathcal{L}_P$ and either $\mathcal{P}$ can spend u by computing a witness within polynomial time, or u contains $\mathcal{P}$'s collateral. A honest party can lose access to their coins either (1) by having a UTxO and its associated collateral be locked within $\mathcal{L}_P$ indefinitely such that the party cannot move it to $\mathcal{L}_0$, $\mathcal{L}_1$ or $\mathcal{L}$, or (2) if a party without consent to spend can use the UTxO as input in a transactions spending it in the process. Note that since by construction, if a UTxO is spent in any way on $\mathcal{L}_P$ its collateral is moved to another UTxO. Thus, to show balance security for collateral we need to show that case (1) cannot occur. To show balance security for the coins in u itself we have to show that (1) cannot occur and that (2) can only occur with negligible probability. Since $\mathcal{L}_P$ has the liveness property, case (1) cannot occur. In the following, we assume that there exists a computationally polynomially bound party that attempts to spend u without receiving consent. This requires creation of a witness for u. As they have no consent to spend the UTxO they cannot compute a witness within polynomial time, thus the probability they can spend it is negligible.

7 Conclusion

In this work we presented a means for parties on two Layer-2 ledgers to interact with another ad-hoc and with little in advance setup. We showed properties *balance security*, *liveness* and *offchain efficiency* hold in the presence of a malicious adversary corrupting all but one parties. While we presented only the core of the construction, we proposed multiple potential extensions as improving efficiency through batching of transactions, recovery from disputes, creation of virtual ledgers and connecting more than two Layer-2 ledgers. We argue that the construction can be used as a framework for secure interaction and individual uses cases can be optimized to facilitate low-overhead interactions.

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. Goldwasser, S., Micali, S., Rivest, R.L.: A "paradoxical" solution to the signature problem. In: Proceedings of the 25th Annual Symposium OnFoundations of Computer Science, 1984. p. 441448. SFCS '84, IEEE Computer Society, USA (1984). https://doi.org/10.1109/SFCS.1984.715946, https://doi.org/10.1109/SFCS.1984.715946
10. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281308 (apr 1988). https://doi.org/10.1137/0217017, https://doi.org/10.1137/0217017
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. Jourenko, M., Larangeira, M., Tanaka, K.: Interhead hydra: Two heads are better than one. In: The 3rd International Conference on Mathematical Research for Blockchain Economy (2022)
15. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Theory of Cryptography Conference. pp. 477–498. Springer (2013)

16. Kiayias, A., Litos, O.S.T.: A composable security treatment of the lightning network. IACR Cryptology ePrint Archive **2019**, 778 (2019)
17. 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)
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. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
20. PDecker, C., Russel, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin. See https://blockstream.com/eltoo.pdf (2017)
21. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. See https://lightning. network/lightning-network-paper. pdf (2016)
22. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 347–356 (2019)