Scalable and Lightweight State-Channel Audits

Christian Badertscher[1], Maxim Jourenko[⋆3], Dimitris Karakostas[2(✉)], and Mario Larangeira[3]

[1] IOG, `christian.badertscher@iohk.io`
[2] University of Edinburgh, `dkarakos@ed.ac.uk`
[3] Tokyo Institute of Technology & IOG,
`jourenko.m.ab@m.titech.ac.jp`,
`mario{.larangeira@iohk.io,@c.titech.ac.jp}`

Abstract. Payment channels are one of the most prominent off-chain scaling solutions for blockchain systems. However, regulatory institutions have difficulty embracing them, as the channels lack insights needed for Anti-Money Laundering (AML) auditing purposes. Our work tackles the problem of a formal reliable and controllable inspection of off-ledger payment channels, by offering a novel approach for maintaining and reliably auditing statistics of payment channels. We extend a typical trustless Layer 2 protocol and provide a lightweight and scalable protocol s.t.: (i) every state channel is provably auditable w.r.t. a configurable set of policy queries, s.t. a regulator can retrieve reliable insights about the channel; (ii) no information beyond the answers to auditing queries is leaked; (iii) the cryptographic operations are inexpensive, the setup is simple, and storage complexity is independent of the transaction graph's size. We present a concrete protocol, based on Hydra Isomorphic State Channels (FC'21), and tie the creation of a state channel to real-world identifiers, both in a plain and privacy-preserving manner. For this, we employ verifiable credentials for decentralized identifiers, specifically verifiable Legal Entity Identifiers (vLEI) that increasingly gain traction for financial service providers and regulated institutions.

1 Introduction

Layer 2 protocols enable interactions between parties without the immediate aid of a public ledger [26,21]. This family of protocols includes payment [20], state [16], and multiplayer channels [10,15] which require only an initial step that locks the funds of the parties in the Layer 1 ledger, as well as networks [31]. Off-chain protocols ease the burden of a blockchain system and enable the generation, and verification of transactions in a low-cost and fast manner, which depends exclusively on the network delays instead of the ledger's confirmation time. As a consequence, payment channels are often the go-to scalability solution in large-scale deployments of cryptocurrencies.

⋆ This work was supported by Input Output Global and JST CREST JPMJCR2113, Japan.

Notably, the main efficiency benefit of Layer 2 protocols, namely the ability to transact in bulk off-chain, also makes them opaque. It is known that the Layer 1 ledger is oblivious to the Layer 2 transactions since none of them is registered in public. This feature introduces a significant burden to regulated institutions and service providers, like banks and government bodies in the presence of auditing according to Anti-Money Laundering (AML) policies.

For a more concrete example, consider that a typical question that authorities ask is whether two parties transacted during a certain period of time and whether the exchanged funds were above some legal threshold. It is trivial to answer such questions in centralized systems and, thus they are part of everyday activities of a compliance department. In practice, AML, Know-Your-Costumer (KYC), and Anti-Terror regulation techniques crucially rely on transaction monitoring. The discussion around Central Bank Digital Currencies (CBDC) [17,24,18] strongly suggests that regulation is a major concern and a current topic of discussion in the industry and academia. Here, the off-chain setting poses a significant gap in addressing such concerns, especially regarding privacy and regulatory compliance, and the topic is currently actively discussed [6].

Evidently, in the presence of digital currencies and smart contracts, new tools are required to answer questions a regulator might pose. Without such tools, an auditor would need to reliably obtain essentially all off-chain transactions which is not only impractical, but goes against the scalability improvements offered by Layer 2 protocols for its users. To the best of our knowledge, selective disclosure capabilities, which are able to answer policy queries, have not been suggested for Layer 2 protocols. Consequently, it is unclear how off-chain protocols can be used efficiently and effectively in a regulatory compliant manner. The existing gap was further identified in a joint report of the ECB and the Bank of Japan [36, §4.2.3], as well as recent Systematization of Knowledge (SoK) work [11] as follows: *"Is auditing of transactions that do not appear in the ledger (or happen "off the chain") possible?"* To the best of our knowledge, our work is the first attempt to formalize the problem statement and close this gap with a positive result.

While our solution puts forth a generic approach to the problem that can be applied to most Layer 2 systems, we offer a first formal result by enhancing the Hydra Protocols [10,28,27]. The Hydra Protocol [10] is a recent generalization of multiparty state channels with smart contract capabilities. In Hydra, a group of parties initiate an *isomorphic* off-chain channel, the "head". Isomorphism ensures that the head's state is enforceable in the Layer 1 ledger, s.t. the head can be used as a ledger on its own, i.e., detached from the consensus protocol. Eventually, a head is closed by replicating the head's state to the ledger seamlessly, thus Hydra puts forth a computational layer that allows the concurrently creation of multiple heads with a well defined computational model, i.e., a state machine.

State-Channel Audit Basic Requirements. State channel audits are an extremely broad goal, but the general approach is to allow regulated institutions to (1) perform bookkeeping efficiently and (2) interact with (potentially) identifiable participants in a way that the obtained statistics are provably sound.

Relevant statistics and policies. In the context of transaction systems, an individual participant (such as a financial service provider) will be subject to audits of several types, where an authority or regulator $\mathcal{R}$ defines the set of statistics that a participant is required to maintain. We call those statistics "policies" and denote the *policy set* by $\Im$. Typically, a regulated institution needs to identify other participants or be assured they are identifiable (KYC). There are also additional prototypical statistics, e.g., that have been identified as relevant in the context of Central Bank Digital Currencies (CBDCs) [25,17,5]: (1) Have parties (P_1, P_2) transacted directly? (2) Have (P_1, P_2) transacted more than T_{tx} assets in one transaction? (3) Within a window of N consecutive transactions, have (P_1, P_2) transacted more than T_{tx} assets on aggregate? (4) Did the balance of party P exceed T_{bal} at any point in time? (5) Within a window of t minutes, has P sent/received more than $T_{\mathsf{send}}/T_{\mathsf{recv}}$ assets on aggregate?

Requirements for an audit protocol. An audit protocol is a cryptographic protocol which needs to satisfy several security-relevant features: (1) *Soundness*: no malicious party (incl. the auditor) can forge a response to an audit query w.r.t. the public execution trail of the audit protocol; (2) *Termination*: the audit process should terminate even if either $\mathcal{R}$ or the audited party abort; (3) *Availability*: $\mathcal{R}$ can query any supported policy at any point in time; (4) *Audit Privacy ([36])*: an audit request should not leak more than the answer to the query.

Aside of the above main properties, one might be interested in further requirements (cf. [11]). First, in certain use-cases the audit trail should be *publicly verifiable*, i.e., $\mathcal{R}$ cannot dispute an honest-party's audit results and a malicious user cannot deny how their interactions with an honest authority. Second, at least a subset of the parties should be *identifiable* and connected to a real-world entity. Finally, auditing should not impose high costs, e.g., blockchain fees.

The audit procedure should ideally introduce very little overhead and complexity to existing transaction protocols. Thus, as a design principle, we aim at a solution where any piece of information that is to be additionally published, for the sake of an audit, needs to be succinct (as some data is expected to be on chain). Furthermore, the storage requirement of honest participants should be less than linear on the number of all transactions created in the Hydra head.

Related Work. Auditability of distributed ledgers has been a point of interest for many years. zkLedger [34], PRCash [39], Garman *et al.* [19], PGC [12] are examples of ledgers that, to some degrees, support auditing by design, i.e., they are designed with auditing capabilities in mind. A relevant line of work concerns efficiently proving the ownership of cryptocurrency assets in a privacy-preserving manner. These works, like Provisions [13], are termed "proofs of solvency" and are primarily used by exchanges. Further, in the context of CBDCs, as mentioned above, extensive auditing capabilities are a core design element and has in fact gained a lot of traction recently [30,40]. The setting and the corresponding solutions however depart heavily from simple peer-to-peer transaction systems since a bank is needed to settle transactions, so they do not provide a (lightweight) solution or drop-in replacement for existing payment channel systems to enable auditing capabilities for decentralized currencies.

Contributions and Roadmap. Our work presents an auditability extension to Layer 2 ledgers and introduces a concrete construction based on the Hydra protocol [10]. The extension is lightweight and requires only small additional values to be published on the main chain. First, Section 3 introduces our proposed auditing framework in a generic Layer 2 model. Then Section 4 focuses on the Hydra protocol [10] to make it auditable. We provide formal definitions of the main goals by extending Hydra's security experiment. Our construction enables identifiability of parties via decentralized identifiers (DIDs) [1], which are integrated in a plain and privacy-preserving manner. To connect our design with practice, we explore the verifiable credentials for Legal Entity Identifiers (LEI) issued by GLEIF [3], a widely accepted way to identify regulated institutions or financial service providers in traditional settlement layers. This integrated real-world identification allows to make the realistic assumption that at least one participant (is obliged to) perform the audit reliably. Specifically, if at least one participant is identifiable [36] in a legal jurisdiction accepted by the auditor, where accessibility to crucial information can be achieved through enforcement, they risk legal implications if valid (in the cryptographic sense) but inconsistent audit information about a state channel are observed, i.e., if parties equivocate. Finally, Section 4.3 presents our main result, a Hydra protocol that enables audits between a regulating authority and the users. Our protocol achieves the main requirements of correctness, soundness, and privacy, as well as public verifiability, identifiability, and efficient auditing. Also it is scalable for policies that can be computed by a Turing machine executing an online algorithm in sub-linear space, where each input is evaluated in sub-linear time.

2 Preliminaries

Before we introduce our construction, it is convenient to briefly recall a state channel protocol, the Hydra construction, and decentralized identifiers which are crucial to ground the identity of the protocol players.

2.1 Hydra: Isomorphic State Channels

Hydra [10] is an isomorphic state channel developed for the Cardano [7].[4] Briefly, it allows any set of parties to move part of the ledger's state off-chain,i.e., the "Hydra head" where participants interact directly. The state machine model is inspired by the Extended UTxO model [9] and Chimeric Ledgers [41].

Hydra employs a multi-signature scheme ⟨MS-Setup, MS-KG, MS-AVK⟩, to generate the global setup parameters, key pairs, and aggregate public key. While the Hydra head operates, there exist two core data types. First, a snapshot $\mathcal{U} = \langle s, U, h, T, S, \hat{\sigma} \rangle$, where: i) s is its number which is generated sequentially; ii) U is its corresponding UTxO set; iii) h is the hash value of the UTxO set; iv) T is the set of transactions that relates this snapshot to the previous one; v) S

[4] A detailed description of Hydra is available in Appendix A.

is its array of signatures (a signature accumulator); vi) $\hat{\sigma}$ is the all participants multi-signature of the snapshot. Second, a transaction $\tau = \langle i, \mathsf{tx}, h, S, \hat{\sigma} \rangle$, where: i) i is the index of the party issuing it; ii) tx is the transaction's information; iii) h is the hash value of tx; iv) S is its array of signatures; v) $\hat{\sigma}$ is its multi-signature. $\mathcal{U}.s$ denotes the snapshot's number (similar for all other parameters of a snapshot and transaction).

The Hydra protocol proceeds in phases. Initially, each party generates their key pair and collects their UTxOs. The keys will form the head's aggregate public key, while the UTxOs form the initial UTxO set. After the parties gather, the Hydra Protocol comprises the actions in Table 1.

	Action	Object/State Identifier
Onchain	Initialize	$\mathsf{init} = \langle \mathsf{initial}, vk_{\text{agg}}, h_{\mathsf{MT}}, n, T \rangle$
	Commit	$\mathsf{commit} = \langle v_{\mathsf{com}}, \mathcal{O}_i \rangle$
	Abort	$\mathsf{abort} = \langle \pi_{\mathsf{MT}} \rangle$
	Open head	$\mathsf{collectCom} = \langle \mathsf{initial}, vk_{\text{agg}}, h_{\mathsf{MT}}, n, T \rangle$
	Close head	$\mathsf{close} = \langle \pi_{\mathsf{MT}}, \xi \rangle$
	Contest	$\mathsf{cont} = \langle \pi_{\mathsf{MT}}, \xi \rangle$
	Finalize	$\mathsf{fanout} = \langle \pi_{\mathsf{MT}} \rangle$
Offchain (in-head)	Request	$\mathsf{reqTx} = \langle \tau \rangle$
	Acknowledge	$\mathsf{ackTx} = \langle \tau.h, \sigma_j \rangle$
	Confirm	$\mathsf{confTx} = \langle \tau.h, \hat{\sigma} \rangle$
	Request	$\mathsf{reqSn} = \langle \mathcal{U}.s, \mathcal{U}.T \rangle$
	Ack. snapshot	$\mathsf{ackSn} = \langle \mathcal{U}.s, \sigma_j \rangle$
	Conf. snapshot	$\mathsf{confSn} = \langle \mathcal{U}.s, \hat{\sigma} \rangle$

Table 1. The parameters are: (i) $\mathsf{initial}$ is a state identifier; (ii) h_{MT} is the root of a Merkle tree for the verification keys of all parties; (iii) n is the number of head members; (iv) T is the length of the contestation period; (v) v_{com} is a validating script, which ensures that the outputs are locked to the right instance of the state machine; (vi) π_{MT} the head's identifier; (vii) ξ is information about the head's state (e.g., a snapshot).

The head is initialized when one of the parties publishes on Layer 1 the state identifier, which establishes the head's initial state and parameters. Each party acknowledges this, s.t. the initiator collects the commitments and opens the head. To create a transaction τ, a party P multicasts τ to all head participants, who validate and sign it. State changes are completed with a new *snapshot*, where a "leader" collects all not-yet-snapshot transactions and multicasts them, each party validates and signs them, and the leader broadcasts a multi-signature.

A head is closed by publishing on Layer 1 a confirmed snapshot that parties may contest, e.g., if incorrect. After the contesting period ends, a finalizing transaction on Layer 1 allows each party to redeem their assets from the head.

Finally, the EUTxO model was extended in the multi-asset EUTxO$_{ma}$ ledger [8], which enables token bundles. These bundles store both a ledger's native currency and fungible and non-fungible tokens. A relevant application for our setting is state thread tokens, that is non-fungible tokens stored within a state machine's assets, which we will use in the context of vLEIs.

2.2 Decentralized Identifiers and vLEIs

Decentralized Identifiers (DIDs), which recently achieved the status of W3C recommendation [1], are reminiscent of traditional digital certificates, e.g., X.509. They aim to be highly dynamic, configurable objects, suited for decentralized applications. In its basic form, a DID structure contains information associated to an entity that enables performing cryptographic operations like signing. A DID is always connected to a particular *method* protocol, that defines how a DID is created, where the associated cryptographic material (DID document) is stored, how it is updated or revoked, and how to retrieve the document given an identity. Our work assumes a prototypical and minimal set of features. A DID object did has the form `did:method:id`, that is a pointer to access public key material relevant to this DID. `method` specifies the method and `id` is typically the hash of the DID's (master) verification key; $\mathsf{did}.vk$ denotes the result of resolving did to obtain its verification key.

A verifiable credential (VC) [2], vc, is a signed statement by a (trusted) credential issuer, that asserts certain claims about a subject $\mathsf{vc.sub}$ identified by a did. Here, we are interested in a VC called vLEI [3] that is of a particular simple type. The claim appearing in a vLEI is just the LEI, the Legal Entity Identifier, issued by GLEIF [3]. Recall that the LEI is not a new invention [4], but the initiative to cast it as VC is very recent. In more detail, a vLEI can be abstracted as the tuple $\mathsf{vc} = (\mathsf{sub}, n, \sigma)$ denoting the subject (the DID of the organization for which the LEI is issued), the LEI-number, and the signature of the GLEIF-accredited LEI issuer, respectively. We assume that such an accreditation happens within a smart contract maintained by GLEIF for example.

3 Auditable State Channels

This section reviews UTxO ledgers and Layer 2 protocols and formally presents the audit extension framework and its security properties.

The UTxO Ledger. In the Unspent Transaction Output (UTxO) model of blockchains, such as Bitcoin [33], the ledger's state Σ is a set of UTxO which are tuple of form (b, ν, δ) where $b \in \mathbb{N}$ is an amount of coins, $\delta \in \{0,1\}^*$ and $\nu \in \{0,1\}^*$ is a verification script such that the UTxO can be spent by a transaction if presented a witness $w \in \{0,1\}^*$ where $\nu(w, \delta) = 1$. A transaction tx of form $(\mathsf{In}, \mathsf{Out})$ can be used to change the ledger's state by specifying a set of UTxO Out as well a list of UTxO In. A transaction can only modify a ledger's state if all UTxO in In are within Σ, i.e. $\mathsf{In} \subseteq \Sigma$, and otherwise is called *invalid*. Two transactions tx_0 and tx_1 are *conflicting* if $\mathsf{tx}_0.\mathsf{In} \cap \mathsf{tx}_1.\mathsf{In} \neq \varnothing$. We denote following state transition function $\circ$ on sets of UTxO $\mathcal{O}$ where $\circ : \mathcal{O}, \mathcal{X} \to \mathcal{O}$ where $\mathcal{O} \circ \mathsf{X} = \mathcal{O} \cup (\bigcup_{\mathsf{tx}:\mathcal{X}} \mathsf{tx.Out}) \setminus (\bigcup_{\mathsf{tx}:\mathcal{X}} \mathsf{tx.In})$ or $\mathcal{O} \circ \mathsf{X} = \perp$ if $\mathcal{X}$ contains either conflicting or with respect to $\mathcal{O}$ invalid transactions. Then, when the transaction $\mathsf{tx} = (\mathsf{In}, \mathsf{Out})$ is applied to the ledger, it's state changes to $\Sigma' = \Sigma \circ \{\mathsf{tx}\}$. Note that while the Extended UTxO model [9] (EUTxO) explicitly include δ in its definition, Bitcoin's [33] UTxO can embed δ within their script ν.

Layer 2 Ledgers. Similar to Jourenko et al. [27] we assume the existence of a family of protocols that construct Layer 2 ledgers $\mathcal{L}^2$. Moreover, we define their security properties analogous to Hydra [10]. Layer 2 Ledgers are structures created on a physical ledger $\mathcal{L}$, such as Bitcoin, are executed by $n \geq 2$ parties $\mathcal{H}$ where $H_{cont} \subseteq \mathcal{H}$ are honest parties. It function as follows: (1) A Layer 2 ledger maintains a set of UTxOs as state Σ^2, (2) UTxO can be moved in-between the physical ledger's state Σ and Σ^2, (3) the parties operating $\mathcal{L}^2$ have means to modify Σ^2 through a set of – potentially implicit – transactions $\mathcal{X}$ and (4) when performing m transactions on $\mathcal{L}^2$ at most a sublinear amount of transactions are committed on $\mathcal{L}$. Note that concrete implementations can restrict moving UTxO between $\mathcal{L}$ and $\mathcal{L}^2$ to setup and tear-down of the protocol. Moreover, transactions on $\mathcal{L}^2$ can have lower expressiveness than those on $\mathcal{L}$. We observe that a wide variety of protocols fall into this family of constructions such as Bitcoin's Lightning payment channels [35] where $n = 2$ parties lock some of their UTxO on $\mathcal{L}$, effectively moving them to $\mathcal{L}^2$ during the opening of the channel to perform a potentially arbitrary amount of transactions, which are limited to payments, using the coins locked in these UTxOs. Lastly the parties will retrieve their coins by closing the channel and moving two UTxO into $\mathcal{L}$ that represent the channels latest balance distribution. On the other side, the Hydra protocol [10] allows for the creation of a multi-party state channel among an arbitrary amount of parties that holds a set of UTxO as state explicitly and where transactions can modify the Hydra channel's state with the same expressiveness as on the physical ledger $\mathcal{L}$.

A Protocol Framework for Trustless Layer 2 Ledgers. We assume that any transaction that modifies the state of $\mathcal{L}^2$ has to be acknowledged by all parties $\mathcal{H}$, so we consider the following framework. At any point, a party can broadcast $(\mathsf{reqTx}, \mathsf{tx})$ to all other parties requesting confirmation of transaction tx. If a party $\mathcal{P}_i$, $0 \leq i \leq n$ receives $(\mathsf{reqTx}, \mathsf{tx})$ then $\mathcal{P}_i$ acknowledges it if it is neither invalid nor conflicting with any previously acknowledged transactions. If tx is acknowledged, $\mathcal{P}_i$ broadcasts $(\mathsf{ack}, \mathsf{tx})$ to all parties in $\mathcal{H}$. In the following, let $\hat{S}_i$ be the set of all transactions for which party $\mathcal{P}_i$ broadcasts $(\mathsf{ack}, \mathsf{tx})$ and let $\bar{C}_j$ be the set of all transactions for which party $\mathcal{P}_j$, $0 \leq j \leq n$ received $(\mathsf{ack}, \mathsf{tx})$ from all parties in $\mathcal{H}$. This framework does not limit existing protocols and instead covers what – potentially implicitly – is already done in existing trustless Layer 2 ledger constructions like Payment Channels or Hydra [10].

The extension to an auditable Layer 2 ledger can be simply formalized at this abstract layer: given a designated auditor $\mathcal{R}$ and the set of admissible policy queries $\Im$, a set of functions on the (confirmed) transaction $\bar{C}$, the auditor is allowed to ask queries of the form (query, Q) upon which it is allowed to interact with a participant P_i of its choice, and the auditor finally produces the functional output $(\mathsf{auditRes}, Q, v)$. Furthermore, the auditable Layer 2 ledger must support a validity predicate $\mathcal{V}$ to judge the validity of an auditor's output in an execution.

Security Properties. We define security of Layer 2 ledgers analogous to Hydra [10]. In the following let the UTxO set $\mathcal{O}_0$ be the initial state of $\mathcal{L}^2$, i.e. Σ^2 and $\mathcal{O}_{\mathsf{final}}$ be the final state of $\mathcal{L}^2$ upon its tear-down. For audits, we state the most fundamental property, soundness, and defer the definition of further properties (including privacy) directly to our concrete Hydra realization.

A *trustlessly* secure Layer 2 ledger has following properties [10].

- Consistency ($\mathcal{L}^2$): For all, i, j, $\mathcal{O}_0 \circ (\bar{C}_i \cup \bar{C}_j) \neq \bot$, i.e., no two uncorrupted parties see conflicting transactions confirmed;
- Liveness ($\mathcal{L}^2$): For any transaction tx input via $(\textsc{new}, sid, tx)$, the following eventually holds: $tx \in \cap_{i\in[n]}\bar{C}_i \vee \forall i : \mathcal{O}_0 \circ (\bar{C}_i \cup \{tx\}) = \bot$, i.e., every party will observe the transaction confirmed or every party will observe the transaction in conflict with his confirmed transaction;
- Soundness ($\mathcal{L}$): $\exists \tilde{S} \subseteq \cap_{i\in\mathcal{H}}\hat{S}_i : \mathcal{O}_{final} = \mathcal{O}_0 \circ \tilde{S}$, i.e., the final UTxO set results from a set of seen transactions;
- Completeness ($\mathcal{L}$): For $\tilde{S}$ as above, $\cup_{P_i\in H_{cont}}\bar{C}_i \subseteq \tilde{S}$, i.e., all transactions seen as confirmed by an honest party at the end of the protocol are considered;
- Audit Soundness: For any request (query, Q) by $\mathcal{R}$ towards any party P_j holding confirmed transaction set $\bar{C}_j$ as above, the auditor's output $(\mathsf{auditRes}, Q, v)$ satisfies $v.val = Q(\bar{C}_j)$ whenever $\mathcal{V}(v, \bar{C}_j) = 1$;
- Audit Correctness: For any request (query, Q) by $\mathcal{R}$ towards any honest party P_j holding confirmed transaction set $\bar{C}_j$ as above, the auditor's output $(\mathsf{auditRes}, Q, v)$ satisfies $\mathcal{V}(v, \bar{C}_j) = 1$;
- Audit Privacy: This is defined through a game played by two adversaries $\mathcal{A}_\tau$ and $\mathcal{A}_{Aud}$ that do not share any private information and a challenger $\mathcal{C}$. Let $\mathcal{O}^0$ and $\mathcal{O}^1$ be two UTxO sets generated by two executions of a trustless Layer 2 ledger $\mathcal{L}_0^2$ and $\mathcal{L}_1^2$ on a common initial UTxO set $\mathcal{O}_0$ chosen by $\mathcal{A}_\tau$. The game is executed by having $\mathcal{C}$ pick $b_C \in \{0,1\}$ and $\mathcal{A}_\tau$ submit a sequence of tuples $(b_\mathcal{A}, \tau)$ upon which τ is applied on $\mathcal{L}^2_{b_\mathcal{A}}$. At any point $\mathcal{A}_{Aud}$ can submit (query, Q) upon which $\mathcal{C}$ responds with $(\mathsf{auditRes}, Q, v_b)$ for both $\mathcal{L}_b^2$, $b \in \{0,1\}$ respectively, if $v_0.val = v_1.val$ and with $\bot$ otherwise. Finally, $\mathcal{A}_{Aud}$ submits b_{Aud} and wins if $b_{Aud} = b_C$. The advantage of the attacker is defined as $|\mathsf{Pr}(b_{Aud} = b_C) - \frac{1}{2}|$ and a protocol has audit privacy if the advantage of any efficient adversary is negligible.

4 Auditable Hydra Isomorphic State Channels

In this section we provide a concrete instantiation of an auditable Layer 2 protocol. Our proposal extends Hydra [10] (cf. Section 2.1), to introduce audit capabilities, and makes use of Decentralized Identifiers (DID) (Section 2.2), to enable meaningful auditing based on real-world identities.

4.1 Auditable Hydra

We now describe the auditing extension to the Hydra Protocol (Table 2). Auditing is performed between a special party, the regulating authority $\mathcal{R}$, and the set of the Hydra head's participants $\mathbb{P}$.

Parameters. The auditing protocol supports a set of policies $\Im$, each of which takes as input a transaction graph, given by any participant, or subset of $\mathbb{P}$, in the

	Action	Object/State Identifier
Auditing onchain	Initialize	$\mathsf{auditInit} = \langle \Im, vk_{\mathcal{R}}, \mathcal{T}, \mathsf{t}_{\text{lei}} \rangle$
	Commit snapshot	$\mathsf{commitSn} = \langle C_G \rangle$
	Close head	$\mathsf{closeHead} = \langle C_G \rangle$
Auditing offchain	Request	$\mathsf{reqAudit} = \langle Q, \mathbb{P}_Q \rangle$
	Respond	$\mathsf{resAudit} = \langle \pi_P \rangle$
	Validate	$\mathsf{valAudit} = \langle b^P_\pi, d^P_Q \rangle$

Table 2. In addition to the parameters of Table 1, the auditable Hydra protocol is parameterized by the following: (i) $\Im$ is a set of auditable policies Q; (ii) $vk_{\mathcal{R}}$ is the public key of the auditing authority $\mathcal{R}$; state thread token t_{lei} via which the smart contract of the vLEI issuer(s) can be referenced to verify credentials; (iii) C_G is a commitment to a representation of the transaction graph G, honestly-generated by function $\mathsf{Com}(\cdot)$; (iv) π_P is a proof of a party's response to an auditing request, honestly-generated by function $\mathsf{ProofGen}(\cdot)$ (given a policy in $\Im$ and a transaction graph G); The verification generates two values, a bit b^P_π and the information d^P_Q.

head. Each request can be addressed, from the $\mathcal{R}$, to any head participant even after the head is already closed. Moreover, the protocol requires pre-negotiated timing parameters $\mathcal{T}$ that specify an upper limit on the rate of transactions, which is used to implement time based policies as further described in Section 4.5. Finally, the Hydra participants must agree on the vLEI credential issuer, in the form of the unique state token associated to the issuer-maintained contract on the main chain t_{lei}. That is, the issuer-governed list of accredited public keys that certify verifiable credentials (in this case simple vLEI identifiers).

Initializing. We extend the initialization procedure in a non-trivial way, to achieve query support as well as building trust in the form of identification. When the head is initialized, each participant $P \in \mathbb{P}$ is given a set of policies $\Im$ and the public key of $\mathcal{R}$ $vk_{\mathcal{R}}$. These parameters are used for auditing requests, when $\mathcal{R}$ queries a head participant about policies in $\Im$. This step can be combined with the original Hydra's initializing process, i.e., init, in a straightforward manner.

Recall that each party acknowledges and verifies the initialization by posting a transaction which locks their outputs to head. In an auditable Hydra head, committing UTxOs is extended to give the participant two options: either submit an ordinary commit-transaction or an audit-commit-transaction.

An audit-commit is richer as it contains additional elements. First, aside of the referenced locked UTxO set $\mathcal{O}_i$, a party P_i specifies, in the data field of the transaction's output, a DID identifier did_i and a verifiable credential vc_i. A valid commit transaction must satisfy the property of the standard commit transaction regarding the validity of $\mathcal{O}_i$. Second, the transaction is signed w.r.t. $\mathsf{did}_i.vk$. Third, vc_i is a vLEI credential s.t. $\mathsf{vc}_i.\mathsf{sub} = \mathsf{did}_i$ the credential's signature verifies as valid w.r.t. a public key identified via a valid reference into the data field of an unspent output identified via state-thread token t_{lei}. That is, the state machine maintained by the trusted LEI issuer. In summary, an audit-commit binds a UTxO set not only to one particular Hydra-head, but additionally presents a real-world credential via which the participant can be identified. A natural extension can make this process privacy-preserving, where the party reveals the credential only to the regulator and proves publicly that the regulator can identify them.[5]

As before, the head's initiator collects the commitments and formally opens the head by publishing a transaction with collectCom in the main ledger. The only change here is that this state transition is only successful if at least one committed output is from an audit-commit transaction.

Snapshot commitment and head closing. The head participants regularly commit publicly to the head's snapshot. In this way, the authority gains access to a commitment of the transaction graph. In effect, in this step the participants commit to the version of the transaction history that $\mathcal{R}$ can query. The final such commitment is published upon closing the head, which again can be part of the close step of the original Hydra protocol.

Main chain and on-chain verification. The head's status is kept by the variable η which is part of the state of the state machine (SM), and it is updated by the *onchain verification (OCV) algorithms*, namely Initial, Close and Contest and Final, as in the original Hydra protocol. Here, we add the audit related algorithm ProofGen. Each head participant runs the Prot algorithm representing the head SM, hence the auditable head protocol AHP is the tuple $(\mathsf{Prot}, \mathsf{Initial}, \mathsf{Close}, \mathsf{Contest}, \mathsf{Final}, \mathsf{ProofGen})$. Note that the participants can be asked audit queries, so they are equipped with ProofGen, while only the $\mathcal{R}$ is requested to execute ProofVal, thereby AHP does not contain ProofVal.

Auditing. Each participant is equipped with the algorithm ProofGen, while the $\mathcal{R}$ can execute ProofVal. Respectively for answering audit queries, and checking the validity of the proof. The auditing proceeds in three phases. First, $\mathcal{R}$ makes a request by sending a policy $Q \in \Im$ to a subset of participants $\mathbb{P}_Q \subseteq \mathbb{P}$. Second, each queried participant constructs a response π. If the participant is honest, it constructs the response π by running their respective ProofGen algorithm. Third, $\mathcal{R}$ verifies the responses. Specifically, for every response π it obtains two bits $\langle b_\pi, d_Q \rangle$ by running ProofVal on each reply. If, for some party, the proof bit is false, that is $b_\pi = 0$, then $\mathcal{R}$ outputs this party, signifying that its response was invalid. Additionally, the authority outputs the policy query response d_Q, which corresponds to a response with a valid proof bit.[6]

Smart contracts and addresses. Smart contracts are a challenge for auditability, as party interactions through smart contracts strongly depend on their code. Storing the code and all interactions for auditability would result in a memory cost linear to head's transactions, violating our efficiency requirements. However, not tracking this information would allow parties to trivially avoid auditability by using a smart contract with the sole purpose of forwarding coins to a recipient. Similarly, tracking all newly created addresses within a Hydra head would result in high memory costs, but not tracking them would enable audit evasion. Note that this issue only applies to addresses and smart contracts that are active. Addresses that do not spend coins, e.g., belong to parties ex-

[5] The privacy-preserving variant is deferred to Section 5.1.

[6] As we will show in the security analysis, audit soundness guarantees that, if the proof bit is 1, then the policy response data is correct.

ternal to the Hydra head, are visible on-chain when the respective UTxOs are decommitted. Note that smart contracts can be made uniquely distinct via non-fungible thread tokens [8]. So, to resolve these issues, we employ the following simplification. For each smart contract interaction, one party declares ownership of that contract by signing its thread token. We model the smart contract as a black box s.t. all interactions with it are considered as interactions that its owner party helped to facilitate. This results in a memory cost linear in the number of parties that execute the Hydra head. Similarly, each address is claimed by one party as its alias, s.t. interactions with that address are attributed to that party.

4.2 Security Experiment

We now instantiate the generic security properties outlined in the previous section for the specific case of Auditable Hydra Protocol. Prior to describing the security experiment for AHP and the implied security notions, we present the more detailed execution.

General Setup. To analyze security we can consider an experiment similar to the original Hydra [10]. Once the main chain machine arrives in the final state, the adversary wins if certain conditions are not satisfied. Consider the following random variables in an execution of the auditable head protocol $\mathsf{AHP} = (\mathsf{Prot}, \mathsf{Initial}, \mathsf{Close}, \mathsf{Contest}, \mathsf{Final}, \mathsf{ProofGen})$:

- The set of n parties P_i running the auditable head protocol with the parameters from the setup phase and an initial UTxO set $\mathcal{O}_0$;
- The adversary $\mathcal{A}$ who can see the messages exchanged by the participants P_i and chooses the initial UTxO set $\mathcal{O}_0$;
- The regulator $\mathcal{R}$ which can choose a single policy $Q \in \Im$ and submit it to any P_i with $i \in [n]$;
- The set of (at the time) uncorrupted parties H_{cont} who produced ξ upon close/contest request and ξ was applied to correct the information η;
- The set of corrupted participants $\mathcal{C}$;
- Every party P_i's and performs $\langle vk_i, sk_i \rangle \leftarrow$ MS-KG($\mathcal{S}$), and computes the aggregate signature $vk_{\text{agg}} \leftarrow$ MS-AVK($\mathcal{S}, \underline{vk}$), for the local verification key vector $\underline{vk}$ of the i-th party.

The head-protocol machine Prot has the following environment interface:

- On input of $(\textsc{init}, sid, i, vk_{\text{agg}}, sk_i, \mathcal{O}_0)$, P_i initializes Prot;
- On input of $(\textsc{new}, sid, \tau)$, P_i submits a new transaction τ;
- When the party output $(\textsc{seen}, sid, \tau)$, it announces it has seen τ;
- When the party output $(\textsc{conf}, sid, \tau)$, it announces that in its local view τ has been confirmed;
- On input of $(\textsc{close}, sid)$, it starts head closure and outputs a certificate ξ;
- On input of $(\textsc{cont}, sid, \eta)$, it contests closure and outputs a certificate ξ.

The Experiment. Now, consider the following experiment for the protocol AHP and its OCV functions with audit algorithms.

1. Global Parameters $\mathcal{S} \leftarrow$ MS-Setup, and $\mathcal{S}$ is passed to adversary $\mathcal{A}$ along with the set of policies $\Im$ and the regulator public key $vk_{\mathcal{R}}$;

2. For each party P_i, key material $\langle vk_i, sk_i\rangle \leftarrow$ MS-KG($\mathcal{S}$) is generated, and the vector $\underline{vk}$ of all parties' public keys and vk_{agg} are passed to $\mathcal{A}$, which returns the initial set $\mathcal{O}_0$;
3. Each party P_i's protocol machine is initialized with $(\textsc{init}, sid, i, vk_{\mathrm{agg}}, sk_i, \mathcal{O}_0)$, where $\mathcal{O}_0$ was chosen by $\mathcal{A}$;
4. The adversary now controls inputs to the parties, (e.g., new transactions, close/contest requests) and sees outputs (e.g., seen and confirmed transactions). The following bookkeeping takes place:
   - when an uncorrupted party p_i outputs ξ upon command $(\textsc{close}, sid)$, record $(\textsc{close}, i, \xi)$, for the certificate ξ;
   - when an uncorrupted party p_i outputs ξ upon command $(\textsc{cont}, sid, \eta)$, record $(\textsc{cont}, i, \eta, \xi)$, such that η is the context information.
5. In "parallel" to the above, the experiment sets $\mathcal{C}$, $H_{cont} \leftarrow \emptyset$, and does the following to simulate the main chain:
   - Initialize $\eta \leftarrow (\mathcal{O}_0, 0, \emptyset)$;
   - When $\mathcal{A}$ supplies (i, ξ') : If i is uncorrupted, ξ' gets replaced by the ξ record in $(\textsc{close}, i, \xi)$ and $H_{cont} \leftarrow H_{cont} \cup \{i\}$. Then $\eta \leftarrow \mathsf{Close}(vk_{\mathrm{agg}}, \eta, \xi)$ and $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$ is computed. If Close rejects, everything in this step is discarded and the step is repeated;
   - The adversary gets to repeatedly supply (i, ξ) for $i \notin \mathcal{C}$; if i is uncorrupted, ξ gets replaced by the ξ recorded in $(\textsc{cont}, i, \xi)$ and $H_{cont} \leftarrow H_{cont} \cup \{i\}$. Then, $\eta \leftarrow \mathsf{Contest}(vk_{\mathrm{agg}}, \eta, \xi)$ and $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$ is computed. If Contest rejects, everything in this step is discarded.
   - When $\mathcal{A}$ supplies $\mathcal{O}_{final}$, $b \leftarrow \mathsf{Final}(\eta, \mathcal{O}_{final})$ is computed.
6. $\mathcal{A}$ can submit $(\textsc{audit}, sid, i, \mathsf{reqAudit})$ s.t. $\mathsf{reqAudit} = \langle Q, \mathbb{P}\rangle$ for $Q \in \Im$ and the participants subset $\mathbb{P}$, and receives the reply $(\textsc{AuditReply}, sid, \mathsf{resAudit})$ for $\mathsf{resAudit} = \langle \pi_{\mathbb{P}}\rangle$, s.t. $\pi_{\mathbb{P}}$ is computed by each participant's ProofGen execution, i.e., $\pi_{\mathbb{P}}$ is the $|\mathbb{P}|$-long π proofs tuple;
7. The experiment ends when $\mathcal{A}$ sends $(\textsc{Halt}, sid)$.

Collusion and listener participant. A significant technical challenge is to formally define the properties of audit security properties to cope with the case of total collusion of the head participants. In order to better illustrate this scenario, assume the existence of at least *one single* honest participant. In such a case, any regulator can cross check the replies from all the participants and potentially pinpoint the set of cheaters given their inability to properly reply to the regulator queries. On the other hand, in the case where all the participants of the head are malicious, then they can collude to generate a correct and consistent pair (G^*, C^*_G), of an artificial transaction graph that satisfies all the expectations of the auditor. In such challenging scenario, the regulator/auditor cannot distinguish it from an honest execution of the protocol. The honest participant assumption is a strong guarantee, which does not exist in this case.

Security events. As in the original Hydra, security is captured by a number of events, which correspond to each outlined security property.[7] All events hold

[7] The original Hydra's security events are described in Appendix A.

in the presence of an *active adversary.* We use the standard notation that a transaction τ is valid for a set of UTxO $\mathcal{O}$ if $\mathcal{O} \circ \tau \neq \perp$.

Each participant P_i stores a short representation C_{G_i} and auditing statistics of the graph G throughout the head's life cycle. For every honest P_i, statistics about G_i and C_{G_i} are correct audit information the auditor can expect from an audit query. The existence of an honest participant provides an execution description (G, C_G), which is assumed to be correct and honest. Our formalization assumes a special protocol participant, $P_{\mathcal{L}}$, that is not part of the protocol, i.e., is a "virtual participant", who observes the protocol's execution and records all transactions. Thus, $P_{\mathcal{L}}$ keeps $(G_{\mathcal{L}}, C_{G_{\mathcal{L}}})$ which is considered the true execution of the protocol. Note that such participant does not exist in reality, however it allows us to correctly define and capture the desired audit security properties.

Following, we outline the security properties for the audit hydra head with respect to the *listener participant* $P_{\mathcal{L}}$. Intuitively, correctness guarantees that any honest user's response will be valid, i.e., $\mathcal{R}$ will accept its proof ($b_\pi^{P_i} = 1$). Also soundness guarantees that, if the regulator accepts a party P_i's proof as correct ($b_\pi^{P_i} = 1$), then it should be infeasible for another participant to also present an acceptable proof for a contradictory response to the same policy query.

Definition 1 (Audit Correctness). *Given policy $Q \in \Im$, let $\langle b_\pi^{P_i}, d_Q^{P_i} \rangle$ be the bits output by $\mathcal{R}$ for the response of an honest party P_i; it should hold that $b_\pi^{P_i} = b_\pi^{P_{\mathcal{L}}} = 1$, for the replies with respect to the listener participant $P_{\mathcal{L}}$.*

Definition 2 (Audit Soundness). *Given policy $Q \in \Im$, let $\langle b_\pi^{P_i}, d_Q^{P_i} \rangle$ be the bits output by $\mathcal{R}$ for the response of a party P_i; it should hold that, if $b_\pi^{P_i} = b_\pi^{P_{\mathcal{L}}} = 1$, then $d_Q^{P_i} = d_Q^{P_{\mathcal{L}}}$, for the replies with respect to the listener participant $P_{\mathcal{L}}$.*

Finally, privacy ensures that $\mathcal{R}$ gains no additional information about the transaction graph, beyond what is revealed by the query answer. We assume $\mathcal{R}$ is an external party to the channel, only observing its public footprint. This assumption is required as Layer 2 transactions are not private by default [22]. Although transaction privacy has been researched extensively for Layer 1 protocols, in Layer 2 some proposals only try to increase unlinkability [23,37], but they are still susceptible side-channel attacks and collusion between certain trusted parties. Therefore, our work aims to maintain the level of privacy that Layer 2 solutions offer while enabling auditing.

Definition 3 (Audit Privacy). *Assume two adversaries that share no private state, but both observe the public trace of the execution, denoted $\mathcal{A}_\tau$ and $\mathcal{A}_{Aud}$. $\mathcal{A}_\tau$ drives the generation of the transaction graph, by submitting transactions and choosing an initial UTxO set, while $\mathcal{A}_{Aud}$ submits audit queries. A challenger $\mathcal{C}$ carries the security experiment described above for two executions w.r.t. two different UTxO sets, $\mathcal{O}_0^0$ and $\mathcal{O}_0^1$. $\mathcal{C}$ picks a random bit b which corresponds to the execution subject to audits.*

The execution proceeds as described above, where $\mathcal{A}_\tau$ controls the UTxO set and transaction generation. For every action $\mathcal{A}_\tau$ performs, it specifies to which UTxO set it corresponds. At any moment $\mathcal{A}_{Aud}$ submits audit queries to $\mathcal{C}$. If the *functional output of the query w.r.t. to both $\mathcal{O}_0^0$ and $\mathcal{O}_0^1$ is the same, $\mathcal{C}$ replies to $\mathcal{A}_{Aud}$ with it; otherwise, if the functional output is different, $\mathcal{C}$ outputs a special value $\perp$. Finally, $\mathcal{A}_{Aud}$ submits a bit b_{Aud}, and wins if $b_{Aud} = b$.*

Privacy is violated if the adversary $\mathcal{A}_{Aud}$ in the above game wins with probability $\frac{1}{2} + \epsilon$ for non-negligible values of ϵ.

4.3 Snapshots and Audit Protocol

We now implement the auditable Hydra protocol following Section 4.1. Our construction assumes that at least one head participant is honest and all policies, for which $\mathcal{R}$ might query the parties, are known beforehand.

Our implementation supports queries with *aggregatable values.* Specifically, consider a transaction graph G and the value v of a specific policy w.r.t. G. Given a new transaction τ that is applied on G, the new policy value should be computable given only v and τ, that is without needing to access G.

Protocol idea. The main idea of our construction (Figure 1) is that participants maintain a set of statistics about the transaction graph, which are sufficient to answer $\mathcal{R}$'s queries. Abstractly speaking, the protocol needs two primitives. First, it needs a (binding and hiding) commitment function, which gets as input a set of values (the statistics maintained about a transaction graph) and outputs a succinct value C_G. Second, it needs a proof scheme. Intuitively, proof generation takes two inputs, a policy $Q \in \Im$ and some witness (the maintained statistics), and outputs a proof π, which contains also the answer to the query. For verification, one must know the commitment C_G, a policy Q, and a proof value π, and the output is a bit indicating the validity of π w.r.t. C_G. From a security perspective, the scheme should guarantee that every honestly-generated proof is valid and that no wrong claims can be proven w.r.t. the committed values.

Construction. The construction relies on a zero-knowledge set (ZKS) [32]. A ZKS is a tuple of algorithms $\langle \mathsf{SetupZKS}, \mathsf{CommitZKS}, \mathsf{QueryZKS}, \mathsf{VerifyZKS} \rangle$. Intuitively, a ZKS enables a party to create (via $\mathsf{CommitZKS}$) a succinct commitment to a set of values $\mathbb{S}$, s.t. they can later prove efficiently (via $\mathsf{QueryZKS}$) inclusion ($k \in \mathbb{S}$) or non-inclusion ($k \notin \mathbb{S}$) statements. Crucially, a ZKS reveals no additional information about the committed set beyond the proven statement. This property is important as the auditor $\mathcal{R}$ should not obtain data about the transaction graph, beyond the information revealed as part of the auditing process (cf. Definition 3). For this reason, primitives like Merkle trees or some accumulators [14] are not suitable, as they don't guarantee the ZK property.

In our construction, each element of the set corresponds to a possible answer to all supported policies, for any subset of parties to which the policy can apply. Note that each policy and each party is uniquely identifiable, so the elements can be ordered in a deterministic manner. We note that a property of ZKS is hiding the size of the committed set. In our application this property is not needed, since the number of parties and the policies are known to $\mathcal{R}$. Therefore, we use the relaxed notion of *nearly* ZKS [29], which preserves the ZKS properties but only leaks the committed set's size. Specifically, we use a nearly ZKS scheme based on polynomial commitments [29] (see Appendix B for more details).

For each policy $Q \in \Im$ exists a function q. This function outputs the answer to the policy query, given a previous answer and a transaction. For example, for the policy "Have P, P' transacted?", $q(b, \tau)$ is 1 if either $b = 1$ or τ transfers funds between P, P'.[8] When a transaction τ is added in the head, each party P updates affected elements of their set. For each element l with a policy function q, P updates its value with $\langle q(l, \tau), Q, \tau.h \rangle$, where $\tau.h$ is the hash of τ. Since transactions are broadcast to all parties, honest parties maintain the same set.

Note on updatable commitments: In the standard KZG setting, each new transaction requires an update on the information kept by the participants, including the regeneration of the commitment. A more efficient alternative is to use updatable variants, e.g., a recent concrete construction by Tas and Boneh [38] which allows efficient updates.

When evaluating policies, the parties involved in τ are looked up in $\mathcal{U}.I$, where addresses and state thread tokens are substituted with parties. Policies have access to both τ and $\mathcal{U}.I$, so they can differentiate whether a party was involved directly through a transaction or facilitated a transaction as owner of a state machine. State machine owners are considered intermediaries of a transaction, s.t. they are always considered to be both sender and receiver of the transaction. While this holds true for transactions that perform a state transition of a state machine where a state thread token is in both the transaction's inputs and outputs, this is not the case for initial and final states. Respectively, when evaluating the policies of transactions that create and close a state machine, the owner is implicitly considered to be both sender and receiver.

When $\mathcal{R}$ makes a policy query Q, an honest participant P retrieves the element l which corresponds to the query's answer,[9] constructs an inclusion proof π_l for l w.r.t. the latest published commitment C_G, and responds with $\pi_P = (l, \pi_l)$. $\mathcal{R}$ then validates the element corresponds to the correct policy and verifies the inclusion proof w.r.t. the latest commitment C_G. If the checks pass, it sets a bit $b = 1$, otherwise it sets $b = 0$, and outputs $\langle b, l \rangle$.

4.4 Security Analysis

We now show that the auditable Hydra protocol (Figure 1) satisfies the properties of Section 4.2.

Since our state-machine extension (Table 2) preserves Hydra's structure and merely includes additional values in initializations and snapshots, Hydra's security guarantees (Appendix A) are preserved, providing the assurance of safety and liveness for snapshots that now also include the audit-related commitment.

Therefore, we can proceed to prove the main audit guarantees. First, Theorem 1 proves that the auditable Hydra protocol satisfies audit correctness. This property is directly inherited from the properties of the original Hydra. Second, Theorem 2 focuses on audit soundness. Our proof relies both on Hydra's properties, namely that all parties observe all transactions, and the binding property

[8] Section 4.5 offers more examples of relevant policies.

[9] For ease of notation, Q contains both the policy query and the parties under question.

Auditable Hydra Protocol π_{Audit}

π_{Audit} keeps the initially empty variables $\mathcal{T}$, which is an array of policy answers, and C, a commitment to $\mathcal{T}$.

Initialize: Upon receiving $\mathsf{auditInit}$ for some set of policies $\Im$, wait for array $\mathcal{T}_L$ from P_L. Then set $\mathcal{T} = \mathcal{T}_L$.
Graph Update: Upon receiving confTx for some transaction τ, do the following. For each element l of $\mathcal{T}$, which corresponds to a policy Q with function q, replace l with $\langle q(l, \tau), Q, \tau.h \rangle$, where $\tau.h$ is τ's hash. Then run *Commit Snapshot* as below.
Commit Snapshot: Upon receiving $\mathsf{commitSn}$, compute $\mathcal{T}$'s commitment C and output $\langle C \rangle$.
Close Head: Upon receiving $\mathsf{closeHead}$, output $\langle C \rangle$.
Audit request: Upon receiving $\mathsf{reqAudit}$ for policy Q, which corresponds to element l of $\mathcal{T}$, create an inclusion proof π_l for l and output $\pi = (l, \pi_l)$.
Audit validation: Upon receiving $\mathsf{resAudit}$ with a proof $\pi = (l, \pi_l)$ for policy Q, do the following checks: (i) verify that the response corresponds to Q; (ii) validate the inclusion proof π_l w.r.t. C. If both checks pass, set $b = 1$, otherwise set $b = 0$. Finally output $\langle b, l \rangle$.

Fig. 1. The Auditable Hydra Protocol.

of the used ZKS primitive. Third, Theorem 3 proves our protocol satisfies audit privacy. The main idea here is that, due to the zero-knowledge property of the ZKS, the auditor $\mathcal{R}$ cannot obtain information about the transaction graph or its statistics, beyond the information revealed by each answered policy query.

Theorem 1 (Audit Correctness). *Given a secure implementation of the Hydra state-machine (Appendix A), the auditable Hydra protocol of Figure 1 satisfies audit correctness (Definition 1).*

Proof. By assumption, a correct and secure realization of the Hydra state machine implies consistency and liveness of the head operations [10]. Specifically, an honest user only accepts confirmed transactions, which are (i) non-conflicting (due to safety) and (ii) the same for all parties (due to liveness). So all honest parties, including the listener participant, construct locally the same set of policy answers and each honest party's response is consistent with the published commitment. □

Theorem 2 (Audit Soundness). *Given a secure implementation of the Hydra state-machine (Appendix A), the auditable Hydra protocol of Figure 1 satisfies audit soundness (Definition 2).*

Proof. Audit soundness follows similarly to consistency. Specifically, since honest parties accept only confirmed transactions and all such transactions are available to all parties, honest parties answer with the same value as the listener in the security experiment (Section 4.2). Additionally, due to the binding property of the ZKS which is used in the protocol, an adversary cannot present a forged inclusion proof for a given snapshot's commitment. □

Theorem 3 (Audit Privacy). *Given a secure implementation of the Hydra state-machine (Appendix A), the auditable Hydra protocol of Figure 1 satisfies audit privacy (Definition 3).*

Proof. First, by assumption the adversary $\mathcal{A}_{Aud}$ (Definition 3) that submits audit queries does not know the transaction graph. This implies that no party that is in the Hydra head colludes with $\mathcal{A}_{Aud}$ (that is, with $\mathcal{R}$), otherwise privacy cannot be guaranteed, since all parties observe all transactions.

Consequently, $\mathcal{A}_{Aud}$ has access only to the following information: i) the sequence of published commitments; ii) the inclusion proof for each answered query; iii) the answer to specific policy queries. First, due to the zero-knowledge (hiding) property of the ZKS commitment, $\mathcal{A}_{Aud}$ cannot infer any information about the values in the set given the published commitment or an inclusion proof. Second, the answer to a policy query is not sufficient to violate audit privacy by definition (cf. Definition 3). Therefore, audit privacy is guaranteed by the construction in Figure 1. □

4.5 Implementation Considerations

Our construction opens various practicality questions.
Termination. The auditing protocol should terminate. This can be achieved via timeouts, so if a party fails to respond within reasonable time then it fails the audit. Similarly, the protocol can define a time period during which the authority can issue policy queries, akin to real-world document retention policies.
Disputes. A dispute resolution mechanism could help in case $\mathcal{R}$ tries to reject an honest party's response. This is enabled via the DID features, by requiring the user to sign their responses, s.t. a corrupted authority could not challenge an honest party's response unless forging their DID signature.
Storage. The storage complexity is proportional to the size of the array. Therefore, for a set of policies $\Im$, each pertaining to a pair of parties, the complexity is $O(|\Im| \cdot n^2)$, where n is the number of the head's participants. Note that the authority $\mathcal{R}$ can make a query for any supported policy at any point in time, after the head's creation. This requirement forces each party to indefinitely keep a local copy of the array. In addition, a party's response contains an inclusion proof, so it is logarithmic on the number of elements.
Costs. The cost of auditing for each participant should be upper-bounded and excess costs, e.g., blockchain fees, should be paid by $\mathcal{R}$. This guarantees that a malicious authority cannot impose heavy costs on a head's participants albeit it limits $\mathcal{R}$, since its budget needs to cover the extra fees for auditing queries. In practice, this can again be guaranteed by using a blockchain, where the authority reimburses the transaction fees that a user pays for replying to a query.

Time Parameter. The protocol requires pre-negotiated timing parameters $\mathcal{T} = (n_t, t_l,\ t_\epsilon)$, where $n_t, t_l, t_\epsilon \in \mathbb{N}$ Here n_t is number of transactions, t_l is a time duration and t_ϵ is the drift of a party's local clock. The value $\mathcal{T}$ is chosen s.t. less than t_n transactions are issued in the head within any time window t_l. This is not intended to limit transaction throughput, but instead an estimate provided by the parties to enable implementation of timing based parameters with time and space complexity independent of the transaction graph's size.

Adjustments to Snapshot Generation. Let $\mathcal{T} = (n_t, t_l, t_\epsilon)$ be the timing parameter provided during protocol setup. We require that each snapshot that is created offchain in a Hydra Head contains a timestamp $t_s \in \mathbb{N}$ and parties reject the snapshot if $|t_s - t_c| \geq t_\epsilon$. Parties maintain a list T which contains entries of form (τ, t) where τ is a transaction and $t \in \mathbb{N}$ is a point in time where $t_c - t_l \leq t \leq t_c$ where t_c is the party's local time. For each new transaction τ to be issued in the head parties refuse including τ in the Hydra head if $|T| = n_t$. Moreover, we modify Hydra's snapshot request from $\mathsf{reqSn} = \langle \mathcal{U}.s, \mathcal{U}.T\rangle$ into $\mathsf{reqSn} = \langle \mathcal{U}.s, \mathcal{U}.T, \mathcal{U}.I\rangle$ where $\mathcal{U}.I = (A, C)$. In the following, let $P_i, 1 \leq i \leq n$ be any party that participates in the auditable Hydra head. Then $A = \{(\alpha, P_i, \sigma_i) | \forall \alpha$ where $\langle \alpha, \theta, H(D)\rangle \in \tau.\mathcal{O}_o \vee \langle \alpha, \theta, H(D)\rangle \in \tau.\mathcal{O}_i$ where $\tau \in \mathcal{U}.T, \tau = \langle [\mathcal{I}], [\mathcal{O}_o]\rangle, \langle \mathcal{O}_i, \mathsf{S},\ D, w\rangle \in \tau.[\mathcal{I}] : \exists!(\alpha, P_i, \sigma_i) \in A\}$ specifies the aliases of all parties, i.e. informally for each address α that is in a transaction's inputs or outputs within a snapshot $\mathcal{U}$, there exists exactly one tuple (α, P_i, σ_i) in A that includes a signature σ_i of P_i and declares α to be an address of P_i. Analogously $C = \{(t_s, P_i, \sigma_i) | \forall$ state thread token $t_s \in \theta$ where $\langle \alpha, \theta, H(D)\rangle \in \tau.\mathcal{O}_o \vee \langle \alpha, \theta, H(D)\rangle \in \tau.\mathcal{O}_i$ where $\tau \in \mathcal{U}.T, \tau = \langle [\mathcal{I}], [\mathcal{O}_o]\rangle, \langle \mathcal{O}_i, \mathsf{S}, D, w\rangle \in \tau.[\mathcal{I}]$ $: \exists!(t_s, P_i, \sigma_i) \in C\}$ declares ownership of CEMs, i.e. informally for each state thread token t_s that is in a transaction's inputs or outputs within a snapshot $\mathcal{U}$, there exists exactly one tuple (t_s, P_i, σ_i) in C that includes a signature σ_i of P_i and declares the CEM that is uniquely identified through t_s to be owned by P_i.

Aggregatable Policy Values Our scheme only supports policies with aggregatable values, s.t. users only keep the array of policy answers, instead of the entire transaction graph. In the following we briefly describe how auditable properties as mentioned in Section 1 can be computed and analyze their computational and memory costs. Note that the aim of utilizing aggregatable values is to render the memory and computational requirements of updating these properties to be independent of the amount of transactions performed. Recall that a party, upon receiving confTx for some transaction τ, updates each element l of $\mathcal{T}$, which corresponds to a policy Q with function q, with $\langle q(l, \tau), Q, \tau.h\rangle \in \{0, 1\}$ where the response to a policy Q is 1 if and only if the answer to that policy is true.

From now we deem that two parties $P_i,\ i \in \{0, 1\}$ transact directly with a transaction $\tau = \langle [\mathcal{I}], [\mathcal{O}]\rangle$ if it contains a UTxO $\mathcal{O} \in [\mathcal{O}]$ where $\mathcal{O} = \langle \alpha, \theta, H(D)\rangle$ and an transaction input $\mathcal{I} \in [\mathcal{I}]$, $\mathcal{I} = \langle \mathcal{O}, \mathsf{S}, D, w\rangle$ where $\mathcal{I}.\mathcal{O}\alpha$ contains one of P_i's addresses (or $\mathcal{I}.\mathcal{O}.\theta$ contains a thread token associated with P_i) and $\mathcal{O}.\alpha$ has an address of P_{1-i} (or $\mathcal{O}.\theta$ contains a thread token associated with P_{1-i}) .

Have parties (P_1, P_2) transacted directly? This can be done by storing one bit value for each pair of parties which is initialized with 0. As soon as P_1 and P_2 are observed to have transacted directly, the value is flipped to 1, i.e., parties have transacted. The computational requirement is constant and the total memory requirements to store this for all party pairs is $\mathcal{O}(n^2)$.

Have parties (P_1, P_2) transacted directly an amount of more than T_{tx} coins in one transaction? This is done analogous to computing whether P_1 and P_2 transacted directly, however, in addition an auxiliary value $c \in \mathbb{N}$ is stored which is initialized with 0 and set to $c' \in \mathbb{N}$ if P_1 and P_2 transact directly an amount of coins $c' > c$. Once $c > T_{\mathsf{tx}}$ is observed, a bit value storing the answer to this property is set to 1. As above the computational requirement for updating this value is constant and the total memory requirements to store this for all pairs of parties is $\mathcal{O}(n^2)$.

Within every window of N consecutive transactions, have parties (P_1, P_2) transacted an aggregate amount of more than T_{tx} coins? This is done analogous to computing whether P_1 and P_2 transacted directly, however, in addition an auxiliary list of numbers $c_0, \ldots, c_N$ are stored, together with a pointer value $p \in \{0, 1, \ldots, N\}$. The entries in the list as well as p are initialized to 0. Once two parties are observed to transact directly a value of $c \in \mathbb{N}$ coins the value c_p is set to c and p is set to $(p + 1) \mod N$. If $\sum_0^N \geq T_{\mathsf{tx}}$ is observed, a bit value storing the answer to this property is set to 1. The computational cost of updating this value is $\mathcal{O}(N)$ and the total memory requirements to store this for all pairs of parties is $\mathcal{O}(n^2 N)$.

Did the balance of party P exceed T_{bal} at any point in time? This can be done by storing a bit value for each pair of parties which is initialized to 0 if their initial balance at the start of the protocol is T_{bal} and 1 otherwise. If at any point in time the observed total balance of a party exceeds T_{bal} the respective bit value is set to 1.

Does there exist a path of length N and value T_{tx} between P_1 and P_2 in the transaction graph? This can be done by storing a graph with n nodes where each party P is represented by a node v_P and there exists an edge between nodes v_{P_1} and v_{P_2} if parties P_1 and P_2 have transacted directly with another. Note that the amount of edges is up to $\mathcal{O}(n^2)$. Moreover, we store a list D of form $\{d_{P_i,P_j} | i, j \in \mathbb{N}, 0 \leq i, j \leq n, \, i \neq j\}$ which stores the shortest path length between nodes v_{P_i} and v_{P_j}. The graph is initialized to have no edges and D is initialized correspondingly. Once P_1 and P_2 are observed to have transacted directly with another and no edge between v_{P_1} and v_{P_2} exists, we proceed as follows. An edge between v_{P_1} and v_{P_2} is created and Dijkstra's shortest path algorithm is executed twice, to compute the shortest paths from v_{P_1} and v_{P_2} respectively to all reachable nodes and we update D correspondingly for parties v_{P_1} and v_{P_2}. Note that the weight of each edge is implicitly set to 1. Afterwards, for each pair of parties v_{P_i} and v_{P_j} in the graph we evaluate if $d_{\{P_i,P_j\}} \geq d_{\{P_i,P_k\}} + d_{\{P_k,P_j\}}$ where $k \in \{1, 2\}$ and update D if a shortest path is found in the process. The computational cost is $\mathcal{O}(n^2)$ and the storage requirements to store all required auxiliary data is $\mathcal{O}(n^2)$.

Within any window of t minutes, has party P sent more than T_{send} assets on aggregate? Recall that parties limit the transaction throughput based on the timing parameter $\mathcal{T} = (n_t, t_l)$ such that within any time window t_l at most n_t transactions are included in the Hydra head. Let $t_c \in \mathbb{N}$ be a party's local time at moment of evaluating this policy. Then for each party P we store a list $T_{P,s}$ with entries of form (t, b) where $t \in \mathbb{N}$ is the time stored for the snapshot in which a transaction τ was included in the Hydra head in which P sent $b \in \mathbb{N}$ assets and $t_c - t \leq t \leq t_c$. When evaluating this policy the value $\sum_{(t,b)\in T_{P,s}} b \geq T_{\mathsf{send}}$ the respective bit value is set to 1. Note that the required storage for this policy is $\mathcal{O}(\frac{t n_t n}{t_l})$, i.e. it depends on the timing parameter $\mathcal{T} = (n_t, t_l)$ and the computational complexity is analogous.

Within any window of t minutes, has party P received more than T_{recv} assets on aggregate? This is analogous to the above question on whether P has sent more than T_{send} with the difference of storing how much assets a party has received within a time window T_{recv}.

5 Enhancements

In this section we propose enhancements to the auditable protocol of Section 4. Specifically, we detail how the protocol can support auditing in a privacy-preserving manner, such that parties can be audited without revealing their real-world identity. Next, we discuss how the protocol can be expanded to guarantee consistency even under a setting where all participants in the Hydra head collude.

5.1 Privacy-Preserving Audit-Commits

In the simple version, verifying the statement that a credential vc_i is issued for a DID did_i is based purely on signatures. While this gives an extremely efficient implementation, a financial service provider identified by a LEI does not always want to reveal its presence publicly. For this case, one can validate a non-interactive zero-knowledge argument or proof of knowledge (NIZK). More precisely, we need a NIZK for the following relation that turns the vLEI into an anonymous credential. The statement is given by $x = (vk_{\text{lei}}, \tau, \mathsf{com}, C, pk_{\mathcal{R}}, \texttt{method})$ and the witness is $w = (vk, \mathsf{vc}_{\text{lei}} = (\mathsf{sub}, n, \sigma_{\text{lei}}), \sigma_{\tau}, r_1, r_2)$. The relation must assert that each of the following equations hold:

- $\mathsf{com} = \mathsf{Com}((\mathsf{sub}, n); r_1)$
- $C = \mathsf{Enc}_{pk_{\mathcal{R}}}((\mathsf{sub}, n); r_2)$
- $\mathsf{Verify}(vk_{\text{lei}}, (\mathsf{sub}, n), \sigma_{\text{lei}}) = 1$
- $\mathsf{Verify}(vk, \tau, \sigma) = 1$
- $\mathsf{sub} = \texttt{did:method:id}$ with $\texttt{id} = H(vk)$ (where H is a public hash function).

The equations express that the party controls a valid DID of a given method by a signature on the transaction, and for which a valid credential has been issued by the vLEI issuer. Using this NIZK, the validity is determined in the expected way.[10] First, aside of the referenced locked UTxO set $\mathcal{O}_i$, a party P_i

[10] We assume that the CRS required for the NIZK is part of the parameter set of the Hydra protocol, and part of the initial agreement of participants. Likewise, we assume that the accepted DID `method` is publicly known, such as KERI for vLEI.

specifies, in the data field of the output of the transaction, the two party specific elements of the statement (com, C) and the proof π of the above relation. Second, a valid commit transaction must satisfy the property of the standard commit transaction regarding the validity of $\mathcal{O}_i$. Additionally, π_i must be a valid proof for the statement $x = (vk_{\text{lei}}, \tau, \mathsf{com}_i, C_i, pk_{\mathcal{R}}, \mathtt{method})$, composed of the two values (com_i, C_i) and the public parameters $pk_{\mathcal{R}}$, vk_{lei} (obtained via reference to state machine identified by t_{lei}). The remaining actions remain the same, in particular, collecting those commitments is now done as described in Section 4.1.

5.2 Consistency in Case of Full Corruption

A design goal of our audit protocol is simplicity, in particular easy deployment on top of any Hydra head implementation. A crucial assumption, backed with real-world identification, is that identified parties are disincentivized to fake the reporting due to legal enforcement. It is still worthwhile to investigate what guarantees we could still obtain if all Hydra head participants colluded.

In that setting it is not clear anymore what the "true" transaction graph looks like, because this notion is not well-defined anymore. The best we can do here is to have parties publicly commit to the transaction graph and prove it, using a SNARK type of proof that the committed statistics in the form of the ZKS is formed correctly based on the committed transaction graph. We further need a sequence of such snapshots on the mainchain for security purposes in order to avoid any sort of equivocation later. Those snapshots are further required to be valid continuations of the state of the state channel and therefore, akin to typical rollup systems, need a second SNARK to prove their validity. We observe that such a solution is much more involved, both in design and deployment, as well as with respect to the cost of performing consistent checkpoints.

6 Conclusion

Our work proposes the first auditable Layer 2 protocol, where a regulator can audit a wide range of policies in a scalable and privacy-preserving manner. We instantiate the protocol as an extension of Hydra state channels [10]. Our scheme is very lightweight and the storage complexity depends only on the number of supported policies and participants and not on the channels' transaction graph.

Our work poses various questions for future work. First, our instantiation assumes a set of pre-defined audit policies. Adding policies on the fly, after the channel is opened, would be particularly useful. Additionally, one could consider other policies, beyond aggregatable (cf. Section 4.5), that could be supported, e.g., via storing a partial graph, and which policies require access to the full graph, thus being impractical. Second, regarding privacy, there are two paths for future work. First, a thorough exploration of audit commits could enhance the scheme's guarantees, as discussed briefly in Section 5.1. Second, existing Layer 2 solutions offer minimal to no privacy guarantees w.r.t. participants in the protocol. An interesting direction is defining auditable privacy-preserving state channel protocols, that maintain a high level of privacy while also being efficient. Third, our scheme assumes at least one honest participant. We briefly discuss (Section 5.2) if auditing is possible when all participants collude, but a more thorough analysis could investigate concrete implementations with such guarantees. Finally, Hydra was recently enhanced with head merging [27], s.t. two groups of participants can join a single head. This introduces auditing challenges, e.g., if the heads support different policies, that merit further consideration.

References

1. W3c recommendation on decentralized identifiers. https://www.w3.org/TR/did-core/ (2022), accessed: 2023-09-15
2. W3c recommendation on verifiable credentials. https://www.w3.org/TR/vc-data-model/ (2022), accessed: 2023-09-15
3. Global legal entity identifier foundation (gleif). https://www.gleif.org/en/about/this-is-gleif (2023), accessed: 2023-09-15
4. Legal identity identifier lookup service. https://www.lei-lookup.com/ (2023), accessed: 2023-09-15
5. Bank, E.C.: Exploring anonymity in central bank digital currencies (2019), https://www.ecb.europa.eu/paym/intro/publications/pdf/ecb.mipinfocus191217.en.pdf
6. Buterin, V., Illum, J., Nadler, M., Schär, F., Soleimani, A.: Blockchain privacy and regulatory compliance: Towards a practical equilibrium. Available at SSRN (2023)
7. Cardano: Cardano. https://cardano.org/ (2023), [Online; accessed 7-March-2023]
8. Chakravarty, M.M.T., Chapman, J., MacKenzie, K., Melkonian, O., Müller, J., Peyton Jones, M., Vinogradova, P., Wadler, P.: Native custom tokens in the extended utxo model. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Applications. pp. 89–111. Springer International Publishing, Cham (2020)
9. 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)
10. Chakravarty, M.M., Coretti, S., Fitzi, M., Gaži, P., Kant, P., Kiayias, A., Russell, A.: Fast isomorphic state channels. In: International Conference on Financial Cryptography and Data Security. pp. 339–358. Springer (2021)
11. Chatzigiannis, P., Baldimtsi, F., Chalkias, K.: Sok: Auditability and accountability in distributed payment systems. In: Sako, K., Tippenhauer, N.O. (eds.) Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12727, pp. 311–337. Springer (2021). https://doi.org/10.1007/978-3-030-78375-4_13, https://doi.org/10.1007/978-3-030-78375-4_13
12. Chen, Y., Ma, X., Tang, C., Au, M.H.: PGC: Decentralized confidential payment system with auditability. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020, Part I. LNCS, vol. 12308, pp. 591–610. Springer, Heidelberg (Sep 2020). https://doi.org/10.1007/978-3-030-58951-6_29

13. Dagher, G.G., Bünz, B., Bonneau, J., Clark, J., Boneh, D.: Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 720–731. ACM Press (Oct 2015). https://doi.org/10.1145/2810103.2813674
14. Damgård, I., Triandopoulos, N.: Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538 (2008), https://eprint.iacr.org/2008/538
15. Dziembowski, S., Eckey, L., Faust, S., Hesse, J., Hostáková, K.: Multi-party virtual state channels. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 625–656. Springer (2019)
16. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: PERUN: Virtual payment channels over cryptographic currencies. Cryptology ePrint Archive, Report 2017/635 (2017), https://eprint.iacr.org/2017/635
17. of England, B.: Central bank digital currency opportunities, challenges and design (2020), https://www.bankofengland.co.uk/-/media/boe/files/paper/2020/central-bank-digital-currency-opportunities-challenges-and-design.pdf
18. G7-UK2021: Public policy principles for retail central bank digital currencies. https://www.mof.go.jp/english/policy/international_policy/convention/g7/g7_20211013_2.pdf (2021)
19. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer, Heidelberg (Feb 2016)
20. Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701 (2016), https://eprint.iacr.org/2016/701
21. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: Off the chain transactions. Cryptology ePrint Archive, Report 2019/360 (2019), https://eprint.iacr.org/2019/360
22. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: Layer-two blockchain protocols. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 201–226. Springer, Heidelberg (Feb 2020). https://doi.org/10.1007/978-3-030-51280-4_12
23. Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In: Network and Distributed System Security Symposium (2017)
24. of International Settlements, B.: Central bank digital currencies: Foundational principles and core features. https://www.bis.org/publ/othp33.htm (2020)
25. of International Settlements, B.: Central bank digital currencies: foundational principles and core features (2020), https://www.bis.org/publ/othp33.pdf
26. Jourenko, M., Kurazumi, K., Larangeira, M., Tanaka, K.: SoK: A taxonomy for layer-2 scalability related protocols for cryptocurrencies. Cryptology ePrint Archive, Report 2019/352 (2019), https://eprint.iacr.org/2019/352
27. Jourenko, M., Larangeira, M.: State machines across isomorphic layer 2 ledgers (2023)
28. Jourenko, M., Larangeira, M., Tanaka, K.: Interhead hydra: Two heads are better than one. In: Pardalos, P., Kotsireas, I., Guo, Y., Knottenbelt, W. (eds.) Mathematical Research for Blockchain Economy. pp. 187–212. Springer International Publishing, Cham (2023)
29. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS,

vol. 6477, pp. 177–194. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_11
30. Kiayias, A., Kohlweiss, M., Sarencheh, A.: PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 1739–1752. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560707
31. Kiayias, A., Litos, O.S.T.: A composable security treatment of the lightning network. Cryptology ePrint Archive, Report 2019/778 (2019), https://eprint.iacr.org/2019/778
32. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: 44th FOCS. pp. 80–91. IEEE Computer Society Press (Oct 2003). https://doi.org/10.1109/SFCS.2003.1238183
33. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
34. Narula, N., Vasquez, W., Virza, M.: zkLedger: Privacy-preserving auditing for distributed ledgers. Cryptology ePrint Archive, Report 2018/241 (2018), https://eprint.iacr.org/2018/241
35. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. See https://lightning. network/lightning-network-paper. pdf (2016)
36. Stella, P.: Balancing confidentiality and auditability in a distributed ledger environment (2020), https://www.ecb.europa.eu/paym/intro/publications/pdf/ecb.miptopical200212.en.pdf
37. Tairi, E., Moreno-Sanchez, P., Maffei, M.: A^{2}L: Anonymous atomic locks for scalability and interoperability in payment channel hubs. Cryptology ePrint Archive, Report 2019/589 (2019), https://eprint.iacr.org/2019/589
38. Tas, E.N., Boneh, D.: Vector commitments with efficient updates. Cryptology ePrint Archive, Paper 2023/1830 (2023). https://doi.org/10.4230/LIPICS.AFT.2023.29, https://eprint.iacr.org/2023/1830, https://eprint.iacr.org/2023/1830
39. Wüst, K., Kostiainen, K., Capkun, V., Capkun, S.: PRCash: Fast, private and regulated transactions for digital currencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 158–178. Springer, Heidelberg (Feb 2019). https://doi.org/10.1007/978-3-030-32101-7_11
40. Wüst, K., Kostiainen, K., Delius, N., Capkun, S.: Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2947–2960. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560617
41. Zahnentferner, J.: Chimeric ledgers: Translating and unifying UTXO-based and account-based cryptocurrencies. Cryptology ePrint Archive, Report 2018/262 (2018), https://eprint.iacr.org/2018/262

A Hydra Details

The Hydra Layer 2 construction [10] was developed for the Cardano blockchain [7]. It is defined as a Constraint-Emitting Machine derived from Mealy Machines [9] and puts forth an isomorphic state channel. Briefly, it allows an arbitrary set of players to take part of the ledger's state off-chain into a so called "Hydra head", i.e., group of participants that interact directly with each other, in the same manner as on Layer 1. In detail, the parties act on the ledger, which is abstracted as a state machine. The state machine model, inspired by the Extended UTxO model [9] and Chimeric Ledgers [41], is as follows.

Definition 4 (Ledger State Machine). *Given a hash function H, the Ledger State Machine is a set of primitives and types such the former is given by*

- *Four strings α, D, θ and w, for respectively the address, data, value/quantity of an asset, and the cryptographic witness;*
- *Script S: a script (program) constructed in the ledger's programming language;*
- *List $[\dots]$: an ordered list of items;*

whereas the types are given by

- *UTxO: A single UTxO $\mathcal{O}$, also named output, is given by the tuple $\langle \alpha, \theta, H(D) \rangle$;*
- *Input: A single input $\mathcal{I}$ is given by the tuple $\langle \mathcal{O}, \mathsf{S}, D, w \rangle$;*
- *Transaction: A transaction τ is given by a set of inputs and an ordered list of outputs, that is $\tau = \langle [\mathcal{I}], [\mathcal{O}] \rangle$;*
- *Ledger: It is modeled as a collection of transactions $[\tau]$, that is $\mathcal{L} = [\tau]$.*

From now we present a brief description of some components of the Hydra Protocol which suffice for our purpose. For a complete description of the protocol we refer the reader to [10].

Hydra is parameterized by the signature algorithms MS-Setup (generate global setup parameters), MS-KG (generate multi-signature key pairs), MS-AVK (generate multi-signature aggregate public key). In addition, while the Hydra head operates, there exist two core data types:

- a snapshot $\mathcal{U} = \langle s, U, h, T, S, \hat{\sigma} \rangle$, where: i) s is its number which is generated sequentially; ii) U is its corresponding UTxO set; iii) h is the hash value of the UTxO set; iv) T is the set of transactions that relates this snapshot to the previous one; v) S is its array of signatures (a signature accumulator); vi) $\hat{\sigma}$ is the all participants multi-signature of the snapshot.
- a transaction $\tau = \langle i, \mathsf{tx}, h, S, \hat{\sigma} \rangle$, where: i) i is the index of the party issuing it; ii) tx is the transaction's information; iii) h is the hash value of tx; iv) S is its array of signatures; v) $\hat{\sigma}$ is its multi-signature.

For completeness, in order to denote the snapshot's number, we use $\mathcal{U}.s$ (similar for all other parameters of a snapshot and transaction).

Party gathering. Prior to initiating the head protocol, each party P_i obtains the global setup parameters as $\mathcal{S} \leftarrow$ MS-Setup and generates their own key pair $\langle vk_i, sk_i \rangle \leftarrow$ MS-KG($\mathcal{S}$). In addition, each party collects their respective UTxO $\mathcal{O}_i$, forming the initial UTxO set $\mathcal{O}_0$. Finally, given the vector $\underline{vk}$ of all parties' public keys, the aggregate public key is generated $vk_{\text{agg}} \leftarrow$ MS-AVK($\mathcal{S}, \underline{vk}$) and published. After the parties gather, the Hydra Protocol comprises the actions in Table 1, and proceeds as follows.

Initialization. To initialize the head, a "head initiator" posts the state identifier init within a transaction on the main ledger, which establishes the head's initial state and parameters. Following, each party acknowledges and verifies the initialization by posting a transaction with a commit state identifier, which locks their outputs to the Hydra head. If all goes well, the initiator collects the commitments and formally opens the head via the publication of a transaction with $\mathsf{collectCom}$ in the main ledger. Otherwise, i.e., if a party fails to commit correctly, the head is aborted, which can be publicly verifiable via a published transaction containing abort in the ledger.

Transaction generation. Once that head is initialized, in order to create a new transaction, a party first creates the τ, which contains the transaction's information (i.e., who sends how much value to whom). Next, it multicasts τ to all head members via the message with the reqTx state identifier. Each member validates τ w.r.t. their local state and, if the validation is successful, sends a message with the ackTx response which contains a signature on τ with their own key. When all members reply with ackTx, the transaction's creator sends confTx to all parties, which contains a multisignature (produced from the vector of each member's individual signature). Finally, upon receiving confTx each party inserts τ to their respective local state.

State change/Snapshot generation. Every head state change is completed only with a new *snapshot*. In order to create a new snapshot, a 3-step process takes place, which is (1) the "snapshot leader" collects all not-yet-snapshot transactions into the to be confirmed snapshot $\mathcal{U}$ along with the snapshot's number. Then (2) it multicasts $\mathcal{U}$ to all head members via messages containing reqSn, and, as long as $\mathcal{U}$ is valid w.r.t. to its local state, each party responds with ackSn which contains a signature. Finally, (3) upon collecting signatures from all members, the leader creates a multisignature and sends the snapshot confirmation confSn to all other parties, which then accept $\mathcal{U}$ as the latest canonical one. Thereby a new state of the head.

Closing. To close the head, any party P publishes a close via a regular transaction on the main ledger, which contains a confirmed, signed snapshot. If the snapshot is not the latest, i.e., if P attempts to omit some transactions by publishing an old snapshot, other parties can contest the closing by publishing the transaction containing cont within a pre-specified period, which has the (correct) latest snapshot (as given by Table 1). Finally, after the contesting period ends, a message that has fanout is published on the ledger, allowing each party to redeem their funds from the Hydra head.

State Thread Token. The EUTxO model was extended [8] with multi-asset support, i.e., the possibility to store token bundles, in the $EUTxO_{ma}$ ledger model. These token bundles not only store a ledger's native currency, but also fungible and non-fungible tokens within a UTxOs θ. A relevant application of this are state thread tokens which are non-fungible tokens stored within a state machine's θ. A token t_s is minted in a CEMs initial state. Whenever a CEM performs a state transition, t_s is moved along to be present in the UTxO that represents the CEMs new state. Lastly, t_s is burned when the CEM terminates. Moreover, we note that as t_s is a unique non-fungible token, therefore it uniquely identifies the CEM which it exists in.

Security Events

Here we review the original Hydra protocol security events, regarding the original construction in [10].

Consider the following random variables:

- $\hat{S}_i$: the set of all transactions tx for which the P_i, while *uncorrupted*, output $(\textsc{seen}, sid, tx)$;
- $\bar{C}_i$: the set of all transactions tx for which party P_i, while *uncorrupted* $(\textsc{conf}, sid, tx)$;
- $\mathcal{H}$: the set of parties that remained uncorrupted.
- Consistency (Head): For all, i, j, $\mathcal{O}_0 \circ (\bar{C}_i \cup \bar{C}_j) \neq \bot$, i.e., no two uncorrupted parties see conflicting transactions confirmed;
- Liveness (Head): For any transaction tx input via $(\textsc{new}, sid, tx)$, the following eventually holds: $tx \in \cap_{i\in[n]}\bar{C}_i \vee \forall i : \mathcal{O}_0 \circ (\bar{C}_i \cup \{tx\}) = \bot$, i.e., every party will observe the transaction confirmed or every party will observe the transaction in conflict with his confirmed transaction;
- Soundness (Chain): $\exists \tilde{S} \subseteq \cap_{i\in\mathcal{H}}\hat{S}_i : \mathcal{O}_{final} = \mathcal{O}_0 \circ \tilde{S}$, i.e., the final UTxO set results from a set of seen transactions;
- Completeness (Chain): For $\tilde{S}$ as above, $\cup_{P_i \in H_{cont}} \bar{C}_i \subseteq \tilde{S}$, i.e., all transactions seen as confirmed by an honest party at the end of the protocol are considered.

B Nearly Zero Knowledge Sets (ZKS)

Our protocol (cf. Section 4) makes use of the KZG nearly ZKS scheme based on polynomial commitments [29], which supports updatable commitments [38]. Therefore, let us first revisit polynomial commitments (Figure 2) and then we include the nearly ZKS scheme, which is based on the $\mathsf{PolyCommit}_{\mathsf{Ped}}$ implementation of the KZG polynomial commitments (Figure 3).

The polynomial commitment implementation $\mathsf{PolyCommit}_{\mathsf{Ped}}$ [29] was shown to be a secure polynomial commitment under the *t-SDH* assumption holds in $\mathcal{G}$, i.e., it satisfies the following properties:

- *Correctness.* Let $\mathsf{PK} \leftarrow \mathsf{Setup}(1^\lambda)$ and $C \leftarrow \mathsf{Commit}(\mathsf{PK}, \phi(x))$. For a commitment C output by $\mathsf{Commit}(\mathsf{PK}, \phi(x))$, and all $\phi(x) \in \mathbb{Z}_p[x]$,
  - the output of $\mathsf{Open}(\mathsf{PK}, C, \phi(x))$ is successfully verified by $\mathsf{VerifyPoly}(\mathsf{PK}, C, \phi(x))$, and,
  - any $\langle i, \phi(i), w_i \rangle$ output by $\mathsf{CreateWitness}(\mathsf{PK}, \phi(x), i)$ is successfully verified by $\mathsf{verifyEval}(\mathsf{PK}, C, i, \phi(i), w_i)$.
- *Polynomial Binding.* For all adversaries $\mathcal{A}$:

$$\begin{aligned} Pr(\mathsf{PK} \leftarrow \mathsf{Setup}(1^\lambda), (C, \langle \phi(x), \phi'(x) \rangle) \leftarrow \mathcal{A}(\mathsf{PK}) :& \\ \mathsf{VerifyPoly}(\mathsf{PK}, C, \phi(x)) = 1 \wedge& \\ \mathsf{VerifyPoly}(\mathsf{PK}, C, \phi'(x)) = 1 \wedge& \\ \phi(x) \neq \phi'(x))& \\ = \epsilon(\lambda).& \end{aligned}$$

KZG Polynomial Commitments

$\mathsf{Setup}(1^\lambda, t)$ generates an appropriate algebraic structure $\mathcal{G}$ and a commitment public-private key pair $\langle \mathsf{PK}, \mathsf{SK} \rangle$ to commit to a polynomial of degree $\leq t$. For simplicity, we add $\mathcal{G}$ to the public key PK. Setup is run by a trusted or distributed authority. Note that SK is not required in the rest of the scheme.

$\mathsf{Commit}(\mathsf{PK}, \phi(x))$ outputs a commitment C to a polynomial $\phi(x)$ for the public key PK, and some associated decommitment information d. (In some constructions, d is null.)

$\mathsf{Open}(\mathsf{PK}, C, \phi(x), d)$ outputs the polynomial $\phi(x)$ used while creating the commitment, with decommitment information d.

$\mathsf{VerifyPoly}(\mathsf{PK}, C, \phi(x), d)$ verifies that C is a commitment to $\phi(x)$, created with decommitment information d. If so it outputs 1, otherwise it outputs 0.

$\mathsf{CreateWitness}(\mathsf{PK}, \phi(x), i, d)$ outputs $\langle i, \phi(i), w_i \rangle$, where w_i is a witness for the evaluation $\phi(i)$ of $\phi(x)$ at the index i and d is the decommitment information.

$\mathsf{VerifyEval}(\mathsf{PK}, C, i, \phi(i), w_i)$ verifies that $\phi(i)$ is indeed the evaluation at the index i of the polynomial committed in C. If so it outputs 1, otherwise it outputs 0.

Fig. 2. KZG Polynomial commitments [29] commits to a polynomial $\phi(x)$ of degree $\leq t$.

- *Evaluation Binding.* For all adversaries $\mathcal{A}$:

$$\begin{aligned}
Pr(\mathsf{PK} \leftarrow \mathsf{Setup}(1^\lambda), (C, \langle i, \phi(i), w_i\rangle), \langle i, \phi(i)', w_i'\rangle) \leftarrow \mathcal{A}(\mathsf{PK}) :&\\
\mathsf{VerifyEval}(\mathsf{PK}, C, i, \phi(i), w_i) = 1 \wedge&\\
\mathsf{VerifyEval}(\mathsf{PK}, C, i, \phi(i)', w_i') = 1 \wedge&\\
\phi(i) \neq \phi(i)')&\\
= \epsilon(\lambda).&
\end{aligned}$$

- *Hiding.* Given $\langle \mathsf{PK}, C\rangle$ and $\{\langle i, \phi(i_j), w_{\phi_{i_j}}\rangle : j \in [1, \mathsf{deg}(\phi)]\}$ for a polynomial $\phi(x) \in \mathbb{Z}_p[x]$ such that $\mathsf{VerifyEval}(\mathsf{PK}, C, i_j, \phi(i_j), w_{\phi_{i_j}}) = 1$ for each j:
  - no adversary $\mathcal{A}$ can determine $\phi(\bar{i})$ with non-negligible probability for any unqueried $\bar{i}$ (computational hiding) or
  - no computationally unbounded adversary $\bar{\mathcal{A}}$ has any information about $\phi(\bar{i})$ for any unqueried index $\bar{i}$ (unconditional hiding).

Nearly ZKS scheme

$\mathsf{SetupZKS}(1^\lambda, t)$ outputs $\mathsf{PK} = \mathsf{Setup}(1^\lambda, t)$. t is an upper bound on the size of the set that may be committed.

$\mathsf{CommitZKS}(\mathsf{PK}, \mathbb{S})$ requires $|\mathbb{S}| \leq t$. Define $\phi(x) = \prod_{k_j \in \mathbb{S}}(x - k_j) \in \mathbb{Z}_p[x]$. Output $C = \mathsf{Commit}(\mathsf{PK}, \phi(x))$. Let $\bar{\phi}(x) \in \mathbb{Z}_p[x]$ be the random degree t polynomial chosen in $\mathsf{PolyCommit}_{\mathsf{Ped}}$.

$\mathsf{QueryZKS}(\mathsf{PK}, C, k_j)$ allows the committer to create a proof that either $k_j \in \mathbb{S}$ or $k_j \notin \mathbb{S}$. Compute $\langle k_j, \phi(k_j), \bar{\phi}(k_j), w_j\rangle = \mathsf{CreateWitness}(\mathsf{PK}, \phi(k_j), \bar{\phi}(k_j), k_j)$.
(i) If $k_j \in \mathbb{S}$, output $\pi_{\mathbb{S}_j} = (k_j, w_j, \phi(k_j), \perp)$.
(ii) If $k_j \notin \mathbb{S}$, create $z_j = g^{\phi(k_j)} h^{\bar{\phi}(k_j)}$ and a ZK proof of knowledge of $\phi(k_j)$ and $\bar{\phi}(k_j)$ in $z_j = g^{\phi(k_j)} h^{\bar{\phi}(k_j)}$. Let $\gamma_j = \langle z_j, \text{ZK proof}\rangle$. Output $\pi_{\mathbb{S}_j} = (k_j, w_j, \perp, \gamma_j)$.

$\mathsf{VerifyZKS}(\mathsf{PK}, C, \pi_{\mathbb{S}_j})$ parses $\pi_{\mathbb{S}_j}$ as $(k_j, w_j, \bar{\phi}(k_j), \gamma_j)$.
(i) If $\bar{\phi}(k_j) \neq \perp$, then $k_j \in \mathbb{S}$. Output 1 if $\mathsf{VerifyEval}(\mathsf{PK}, C, k_j, 0, \bar{\phi}(k_j), w_j) = 1$.
(ii) If $\gamma_j \neq \perp$, then $k_j \notin \mathbb{S}$. Parse γ_j as $\langle z_j, \text{ZK proof}\rangle$. If $e(C, g) = e(w_j, g^{\alpha - k_j})e(z_j, g)$ and the ZK proof of knowledge of z_j is valid, output 1.
Output 0 if either check fails.

Fig. 3. The KZG nearly ZKS scheme based on polynomial commitments [29].