BUNGEE: Dependable Blockchain Views for Interoperability

RAFAEL BELCHIOR, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal and Blockdaemon, Lisboa, Portugal
LIMARIS TORRES, Independent Researcher, Miami, United States
JONAS PFANNSCHMIDT, Blockdaemon, Dublin, Ireland
ANDRÉ VASCONCELOS and MIGUEL CORREIA, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

With the evolution of distributed ledger technology (DLT), several blockchains that provide enhanced privacy guarantees and features, including Corda, Hyperledger Fabric, and Canton, are being increasingly adopted. These distributed ledgers only provide partial consistency, meaning that participants can observe the same ledger differently, i.e., observe some transactions but not others, providing higher levels of privacy to the end-user.

Choosing privacy instead of transparency leads to delicate trade-offs that are difficult to manage during runtime, hampering the development of applications that depend on reasoning about shared state, e.g., asset transfers across blockchains. We propose using the concept of *blockchain view* (view) – an abstraction of the state a participant can access at a certain point to address this problem. Views allow us to systematically reason about either state partitions within the same DLT or an integrated view spanning across several DLTs. We introduce BUNGEE (Blockchain UNifier view GEnErator), the first DLT view generator, to allow capturing snapshots, constructing views from these snapshots, and merging views according to a set of rules specified by the view stakeholders. Creating views and operating views allows new applications built on top of dependable blockchain interoperability, such as stakeholder-centric snapshots for audits, cross-chain analysis, blockchain migration, and combined on-chain-off-chain analytics.

CCS Concepts: • **Security and privacy** → **Privacy-preserving protocols**; **Distributed systems security**; *Information accountability and usage control*; • **Information systems** → *Information integration*

Additional Key Words and Phrases: Distributed ledgers, blockchain view, privacy-preserving interoperability, accountability

ACM Reference Format:
Rafael Belchior, Limaris Torres, Jonas Pfannschmidt, André Vasconcelos, and Miguel Correia. 2024. BUNGEE: Dependable Blockchain Views for Interoperability. *Distrib. Ledger Technol.* 3, 1, Article 7 (March 2024), 25 pages. https://doi.org/10.1145/3643689

This work was developed within the project scope nr. 51 "BLOCKCHAIN.PT - Agenda Descentralizar Portugal com Blockchain", financed by European Funds, namely "Recovery and Resilience Plan - Component 5: Agendas Mobilizadoras para a Inovação Empresarial", included in the NextGenerationEU funding program. This work was also supported by the European Commission through contract 952226 (BIG), and national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and 2020.06837.BD.
Authors' addresses: R. Belchior, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Rua Alves Redol, 9, Lisboa, Portugal, 1000-029 and Blockdaemon, 1 Grant's Road, Lower Mount St. Dublin 2, Dublin D02 HX96 Ireland; e-mail: rafael.belchior@tecnico.ulisboa.pt; L. Torres, Independent researcher, 11011 S.W. 104 Street Miami, FL 33176-3393; e-mail: limarist@gmail.com; J. Pfannschmidt, Blockdaemon, 1 Grant's Road, Lower Mount St. Dublin 2, Dublin D02 HX96 Ireland; e-mail: jonas@blockdaemon.com; A. Vasconcelos and M. Correia, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Rua Alves Redol, 9, Lisboa, Portugal, 1000-029; e-mails: andre.vasconcelos@tecnico.ulisboa.pt, miguel.p.correia@tecnico.ulisboa.pt.

ACM 2769-6472/2024/03-ART7
https://doi.org/10.1145/3643689

1 INTRODUCTION

Blockchains[1] provide trustworthy and transparent services, leveraging a network of mutually untrusting participants. A highly desirable property of DLTs is *consistency* [1]: the guarantee that all honest parties share a common prefix of the blockchain, i.e., they see the same *transactions* registered in the ledger. Based on this property, each ledger holds a single source of truth for all its *participants*: consistency is the foundation of the decentralized trust that DLTs offer.[2] The view concept has its roots in database schema integration and, more recently, in business process view integration [2]. To account for the multitude of business process views, ***business view process integration*** **(BVPI)** studies the consolidation of different views regarding a business process [2, 3]. Business view process integration (BPVI) addresses the challenges of processes that involve several participants with different incentives, alleviating them by merging models that represent a different view of the same model.

Despite the importance of consistency, some permissioned DLTs offer only *partial consistency*, providing a trade-off between transparency and privacy. Partial consistency is a weaker notion of consistency that implies that honest parties can read only subsets of the same global transaction graph, i.e., of the ledger. For every transaction ID a set of parties share, they also agree on the contents and dependencies of such transaction [4]. Partial consistent blockchains are very useful in enterprise-grade applications, where privacy and accountability are paramount and necessary to enforce [5]. Organizations working with DLTs that provide partial consistency have been putting resources in place to enable *interoperability* with other blockchains, following the growing trend in the space to accommodate DLTs offering different properties and features [6, 7].

A problem naturally emerges from a multichain ecosystem. Since participants might have different views of a chain, see different data partitions.[3]

A solution to address interoperability across blockchains is to trust a centralized third party [9], such as a cryptocurrency exchange. However, we have seen that such platforms have been consistently attacked and even defrauded its users (see the FTX crash [10], and many more [11, 12]). Trusting a decentralized protocol could be the way forward. However, many decentralized protocols useful for interoperability have few privacy-protecting mechanisms [7, 13] and are still immature and insecure [14]. However, there are alternatives to conventional decentralized bridge designs. A promising one is systems based on zero-knowledge proofs [15, 16]. Despite having potential to be reasonable long-term solutions, these are two shortcomings: (1) the technology is very recent, and thus it is immature and not hard-tested (e.g., zk-rollup technology [17]) and (2) it requires significant engineering effort, to the point of even creating a new blockchain from scratch (see ZCash, Monero [18]).

We then search for a balanced approach that includes support for both centralized and decentralized protocols. A view offers a stakeholder-centric, generalizable, self-describing commitment to the state of a blockchain, allowing for representing states from different blockchains in a standardized way. Blockchain views are created, processed, and shared by consortium participants that are legally identified and have contractual obligations to follow a protocol, namely *blockchain gateways* [19, 20]. Gateways are trusted systems that run an interoperability protocol [21] and can use blockchain views as proof of state for asset and data transfers. Despite being trusted and therefore suitable for enterprises, several decentralization features in gateways promote accountability among consortium members. In this context, building and analyzing views is important to understand each stakeholder's view of each DLT accurately as a tool for dependable *interoperability* across heterogeneous systems.

[1]We use the terms DLT and blockchain interchangeably. A DLT subsumes a blockchain, i.e., a blockchain is a DLT.

[2]A similar assumption could be extended to business processes, where a single model can capture simple processes. However, different representations of the same process are possible as soon as its complexity increases.

[3]Note that it is different from the traditional database field, where different views exist and are processed according to a different number of methods [8]. In particular, we are interested in the problem of managing views of a decentralized system, where access control and data management (who can see what) are decentralized. This has implications for governance and privacy, as no single authority manages the view creation, processing, and sharing processes. Furthermore, operations in the decentralized database must generate and share proof that such an operation is valid.

Problem Definition and Research Questions

Let us consider a blockchain consortium c made up of n nodes and v different views. The research problem we propose to solve is to find a view v' that is agreed by a subset of n, includes a subset of v, and respects a privacy policy defined in c. To answer this problem, we divide it into several **research questions (RQs)**:

— *RQ 1. How to provide a data format for views that balance the characteristics of centralized and decentralized systems?*
Multiple DLT data formats result from their architecture, consensus, and identity models. Formalizing the blockchain view and related concepts is necessary to clarify data representation across chains. *Motivation*: the absence of standards in the interoperability area is a well-known concern [22–24]. Providing a standardized data format would enable the industry and academia to converge to more efficient design patterns, enhancing organizational and legal interoperability.
— *RQ 2. How to guarantee privacy-preserving properties in blockchain interoperability?*
This contribution addresses that need by creating views that allow one to see a stakeholder's perspective over the entire ledger. However, how do we obtain a holistic view of a DLT providing partial consistency, i.e., combined perspectives of all participants, according to privacy restrictions? We ensure that the view's creation, merging, and processing come with privacy, integrity, and accountability guarantees. *Motivation*: Recent research highlights the need and inexistence of privacy-preserving mechanisms in the cross-chain setting as a priority for next-generation interoperable systems [25–27]. Providing privacy-preserving capabilities for cross-chain solutions would enable a new range of users, including enterprises, governments, and organizations that transact sensitive data.

By answering these research questions, we expect to analyze, model, design, and provide implementation guidelines for systems generating views, making it easier to reason about systems interacting with several blockchains, hence delivering the following contributions:

Contributions

— We define the concept of *blockchain view*. We present a formalization of concepts surrounding the view, rooted in the state abstraction and causality relationships between transactions, states, and views. This formalization is the foundation to specify systems handling views, and also for practical implementations.
— We specify the first view integrator, ***Blockchain UNifier view GEnErator* (BUNGEE)**. BUNGEE is a flexible, modular middleware that sits between the data and the semantic layers of a blockchain, allowing data to be abstracted into different data models and formats. To the best of our knowledge, this is the first time views are used to take stakeholder-specific snapshots of the ledger, allowing for several applications.
— We specify BUNGEE's algorithms to create, merge, and process a view, providing a comprehensive discussion of decentralization, efficiency, and privacy trade-offs.

Paper Outline

This document is organized as follows: Section 2 introduces the background necessary to comprehend this paper. In Section 3 we formalize the blockchain view and related concepts. Next, Section 4 presents BUNGEE. After that, we present a discussion in Section 5. Next, we present the related work, in Section 6. Finally, we conclude the paper, in Section 7.

2 PRELIMINARIES

This section presents the background necessary to understand the paper, along with motivational use cases.

Blockchains Providing Partial Consistency. Blockchains providing partial consistency create partitions over the global state according to some criteria. Private blockchains require their participants to be authenticated and

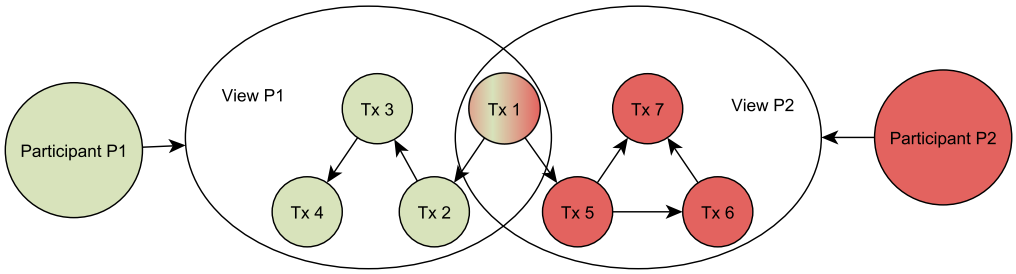

Fig. 1. Two different participant views over the same DLT. Tx stands for a transaction. A green or red labeled transaction is available (read access) to Participant P1 or Participant P2, respectively. Transaction Tx1 is available for both participants.

only expose their content to trusted parties (although those parties do not necessarily trust each other). In private blockchains, different views are common and desirable for privacy reasons [5]. Parties may want to share information with a selected group.

For example, Hyperledger Fabric (Fabric), a private blockchain framework, provides a feature called private data collections. Private data allows sets of participants to hide part of the state they hold, only sharing a hash of that private data as proof of existence [28, 29]. This feature effectively implements partial consistency in Fabric, allowing for the existence of different views. In Corda [30], transactions are ordered as a set of (potentially) disconnected directed acyclic graphs – parties can access certain subgraphs, i.e., Corda provides partial consistency. Other examples exist, such as Quorum [31], IOTA [32], and Digital Asset's Canton [33]. Figure 1 shows a visualization of the concept.

Blockchain Interoperability. The emergence of many blockchains raised the debate about the need for interoperability [7, 13]. Interoperability can be defined as the ability of multiple parties to work together by sharing/exchanging information [34].

The first use case for interoperability is cross-chain state creation, management, and visualization. While some preparatory work has been done [14], it is hard to visualize and reason about private data partitions (different views), not only in the cross-chain setting but also in a single blockchain setting. Blockchain platforms could leverage views to improve view analysis for auditors, cybersecurity experts, and developers. Auditors and cybersecurity professionals can facilitate audits [35] because different data partitions can be analyzed from a specific angle. Developers can gain insight into their applications and processes. Representing on-chain data through a DLT view in multiple chains allows for a visualization of the cross-chain state, making it easier to manage and reason. A specific application could be having one view across multiple Cosmos zones, Polkadot parachains, or Layer 2 solutions (Polygon, Arbitrum, and others, for instance) [36].

The second use case is decentralized application migration. Migration of blockchain-based applications is necessary and increasingly common [13, 37, 38]. Migration allows enterprises to experiment with other DLT infrastructures without the risk of vendor lock-in. The key idea behind application migration is to capture the DLT state relevant to that application (data and functionality) and move it to a different DLT infrastructure. With several views on participants' concerns operating on the source blockchain, one might need to consolidate their diverse views into an integrated view that serves as the foundation of the migration. The integrated view comprises a holistic view of the application's state at the source DLT. This view can then be transferred to the target infrastructure and its functionality (i.e., smart contract migration). We leave the treatment of this interesting problem and its details (for example, how to manage user keys) for future work.

Finally, the third use case is to allow asset transfers between chains [20]. Transferring assets between public DLTs and centralized systems (or private DLTs) is hard because it relies on strong trust assumptions or transparency. An asset transfer is typically implemented by locking an asset in the source chain and unlocking it in the target chain. However, if one of the chains is private (or centralized), such a state is not visible (by design)

[39]. Therefore, decentralized transfers across these types of system rely on proofs (or, rather, a notarization) of the current state of each chain concerning the representation of an asset [40]. Blockchain gateways that perform asset transfers need to translate the state of assets "to some DLT-neutral standard that other gateways can interpret and then hand over to the networks they are acting on behalf of" [41] - the DLT-neutral standard is the view.

Thus, blockchain views can be the bridge that allows decentralized blockchain interoperability across heterogeneous systems by representing such notarization on a public forum [42, 43], with a standardized data format, independent of any specific blockchain implementation. A related use case to this would be to build a cross-chain wallet that, giving a private key, outputs all tokens in all blockchains associated with that wallet. This could be particularly useful for people with LUNA tokens spread across several blockchains, especially after the value of the coin plummeted [44].

3 BLOCKCHAIN VIEWS

This section introduces a running example that applies view integration to the supply chain industry. After that, we formalize concepts related to the view, such as the access point, blockchain view, and view generator.

3.1 Running Example

We present a typical use case on private blockchains, supply chain [45], that benefits from representing the various internal views to an external observer.

A supply chain transfers value between parties, from the raw product (physical or intellectual) to its finalized version. Managing a supply chain is complex because it includes many nontrusting participants (e.g., enterprises and regulators), and implies keeping an audit trail of all operations. As many markets are open and fluid, companies do not take the time to build trust and instead rely on a paper trail that logs the state of an object in the supply chain. This paper trail is necessary for auditability and can typically be tampered with, which leads to the suitability of blockchain to address these problems by monitoring the execution of the collaborative process. Blockchain smart contracts can ensure that the execution of the process complies with defined business rules [46, 47].

Audits inspect the trail of transactions referring to a product's lifecycle. Therefore, different perspectives might need to be analyzed. A challenge naturally emerges: balancing the necessary transparency for audits while maintaining privacy about the transactions across other business partner groups is not trivial. By selectively sharing a common domain, parties can have more efficient processes while performing data-sensitive operations within the same supply chain. A domain is a state shared by parties enrolled in a private relationship.

Let us consider a group of five organizations on a Hyperledger Fabric blockchain that produces, transport, and trade:

— A Supplier, producing goods.
— A Shipper, moving goods between parties.
— A Distributor, moving goods abroad. Buys goods from Suppliers and sells them to Wholesalers.
— A Wholesaler, acquiring goods from the Distributor.
— A Retailer, acquiring goods from shippers and wholesalers.

The Distributor may prefer to make private transactions with the Supplier and the Shipper to keep confidentiality towards the Wholesaler and Retailer (hiding their profit margins). On the contrary, the Distributor may want a different relationship with the Wholesaler. It charges them a lower price than it does with the Retailer (as it sells assets in bulk). The Wholesaler may want to share the same data with the Retailer and the Shipper (because the Wholesaler may charge the Retailers a higher price than the Shipper). We call these private relationships *domains*. The combination of all domains represents the whole ledger.

Domains hold a subset of the ledger that is only accessible by authorized parties. By sectioning the shared ledger, different views on the same blockchain are possible, depending on a stakeholder's participation in a

Table 1. Participant Views on the Supply-chain Blockchain Regarding an Asset with ID = 1

Participant\Domain	d_1	d_2	d_3	
Supplier	ID: 1 Price: 1	ID: 1 Price: hidden	ID: 1 Price: hidden	v_1
Shipper	ID: 1 Price: 1	ID: 1 Price: 2	ID: 1 Price: hidden	v_2
Distributor	ID: 1 Price: 1	ID: 1 Price: hidden	ID: 1 Price: 3	v_3
Wholesaler	ID: 1 Price: hidden	ID: 1 Price: 2	ID: 1 Price: 3	v_4
Retailer	ID: 1 Price: hidden	ID: 1 Price: 2	ID: 1 Price: hidden	v_5

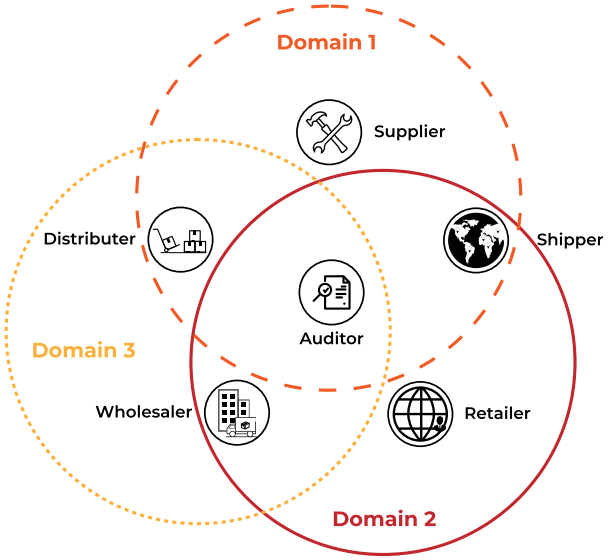

Fig. 2. Different domains on the blockchain supporting the supply chain scenario. For instance, the Supplier has access to domain 1, while the Shipper accesses domains 1 and 2. Access to different domains leads to the creation of different views.

given domain, as shown in Table 1. In this table, the asset ID is one across all domains. However, its price differs across domains, translating into different views. For instance, the Supplier has access to the asset's price on d_1, but only access to its price's hash on d_2 and d_3 (i.e., does not have access to the price on d_2 and d_3. This three-dimensional tuple access-deny-deny corresponds to v_1. Retailers' view, v_5 can see the asset's price only in d_2. The three existing domains (see Figure 2, translate into three different price values for the same item – five participants originate five different views.

Now, assume that we identify each asset tracked in a supply chain by an ID and a price. For the same asset (and thus the same state on the blockchain, as it is uniquely identifiable by its ID), the Distributor-Supplier-Shipper (Domain 1, d_1) has the same view of the price, but the Wholesaler-Retailer-Shipper (Domain 2, d_2) and Distributor-Wholesaler (Domain 3, d_3) and have different views. As every stakeholder has a different combination of the domains that are accessible, the DLT infrastructure yields five different views.

Let us now imagine that an auditor wants to inspect the Distributor's operations regarding an asset. The auditor would retrieve blockchain snapshots in light of each participant's view. After that, the auditor can analyze each view from the perspective of each participant. If a general picture is needed, all views can be merged into an integrated one and jointly analyzed. Since there are different viewpoints, there are different prices for the same object, and different merge procedures are possible. This is not a trivial aspect to deal with, since views can originate from *multiple* decentralized systems, and therefore, aspects such as governance, access control, and privacy-preserving mechanisms are not easily managed by a centralized party.

In particular, the different views are translated into an integrated view that refers only to a consolidated price as a summary of the prices of the different views. The processing and the merging of the views are the consortium's responsibility for managing the blockchain, and thus, several options are possible. We will address this point later in this paper.

3.2 Formalizing Views

This section formally defines the terms necessary for blockchain view integration. We provide the conceptual framework to build programs that can merge blockchain views. The first concept of our framework is the ledger. A ledger is a simple key-value database with two functionalities: `read` and `store`. It supports a state machine that implements a DLT. We define the ledger as follows:

Definition 1 (Ledger). A ledger $\mathcal{L}$ is a tuple $(\mathcal{D}, \mathcal{A})$ such that:

- $\mathcal{D}$ is a database, specifically a key-value store. Each entry in the database is a key-value tuple, i.e., $d \in \mathcal{D} : (k, v)$, where k stands for key and v for value.
  $\mathcal{D}$ has two functions: `read` and writes (storing). `read` returns the value associated with a key, the empty set $\emptyset$ (if there is no value for that key) or an error $\perp$ (if the user does not have access permissions), i.e., $\texttt{read} \rightarrow \{v, \emptyset, \perp\}$. The `store` primitive saves the (k, v) pair in the database, indexed by k, returning 1 if the operation was successful and 0 otherwise, i.e., $\texttt{store} : k \times v \rightarrow \{0, 1\}$.
  The `read` and `store` primitives support the representation of simple UTXO blockchains (e.g., Bitcoin) [48] or more complex ones, an account model (e.g., Ethereum) [49], or others by combining the operations mentioned above (e.g., Hyperledger Fabric [50]).
- $\mathcal{A}$ is an access control list that specifies access rights to read entries from the database. Each entry in the list has the form (p, k). Each entry indicates that the participant p can read the state with key k. A participant p can access a key k when the primitive $\texttt{access}(\mathcal{A}, p, k)$ returns 1, or 0 otherwise.

The simple functionality of the notion of ledger given in Definition 1 allows us to represent Bitcoin [51] as follows: the database (collection of all states) is a list of UTXO entries (states). A UTXO has a unique identifier, the transaction hash, and the state key (we present a simplified version of UTXO). Its value is in the form *(input, output, metadata)*. The input corresponds to a reference to the previous transaction and a key to unlock the previous output to the current input. The output consists of a cryptographic lock and time. Metadata is any other relevant information for a transaction using that UTXO (for example, the timestamp and the fees). Hyperledger Fabric's state is more straightforward to map since it is a key-value store.

We define the entities that can read or write in the ledger by *participants*:

Definition 2 (Participant). A participant $p \in \Upsilon$ is an entity (K_k^{id}, K_P^{id}), capable of reading and writing to a ledger L, where:

- K_k^{id} is a private key. The private key is used as the signing key.
- K_P^{id} is a public key. The public key is used as the verification key.

Participants interact with the ledger via nodes. Nodes are software systems that participate in ledger consensus by aggregating and executing transactions and sending them to other nodes. We introduce the concept of *Access Point* to formalize the relationship between participants and nodes as follows:

Table 2. Ledger l Projections onto all the Participants from the Use Case Depicted in Section 3.1

$d_{\pi_{l,p_n}}$	d_1	d_2	d_3
p_1 = Supplier	s_1	$\overline{s_2}$	$\overline{s_3}$
p_2 = Shipper	s_1	s_2	$\overline{s_3}$
p_3 = Distributor	s_1	$\overline{s_2}$	s_3
p_4 = Wholesaler	$\overline{s_1}$	s_2	s_3
p_5 = Retailer	$\overline{s_1}$	s_2	$\overline{s_3}$
p_6 = Retailer	$\overline{s_1}$	s_2	$\overline{s_3}$

The projection function is a simple read of the database. A state s_i in the green background is a state that a participant can access, whereas a state $\overline{s_i}$ is a state that is not accessible to a given participant.

Definition 3 (Access Point (AP)). An AP ω maps a set of nodes n connected to ledger $\mathcal{L}$ to a set of participants p_n, i.e., $\omega(n) \longrightarrow p_n \subseteq \Upsilon_{\mathcal{L}}$. Conversely, ω^{-1} returns the node set n_p that a participant can access, i.e., $\omega^{-1}(v) \longrightarrow n_p$.

An access point tells us which participants can access the ledger through a specific node. Nodes can access a DLT through the primitive `obtainDLT`. The result of `obtainDLT`$(n_p) = \mathcal{L}_v$, where $\mathcal{L}_v$ is a *virtual ledger*. A virtual ledger only allows the participants to read and write in the ledger according to their permissions (namely, the defined access control list). In more detail, a virtual ledger:

Definition 4 (Virtual ledgers). A virtual ledger $\mathcal{L}_v$ is a projection of a ledger $\mathcal{L}(\mathcal{D}, \mathcal{A})$ in the form $(\mathcal{L}, \mathcal{F}_\pi)$ such that:

- $\mathcal{L}$ is the ledger that provides the database where projections are made.
- $\mathcal{F}_\Pi$, a set of projection functions $\{\mathcal{F}_{\pi_1}, \mathcal{F}_{\pi_2}, \ldots, \mathcal{F}_{\pi_n}\}$ that returns a subset d_π of the database $\mathcal{D}$ from $\mathcal{L}$, i.e., $\mathcal{F}_\pi \in \mathcal{F}_\Pi$: $\mathcal{L}_\mathcal{D} \times \mathcal{L}_\mathcal{A} \times p \rightarrow \{\emptyset, d_\pi\}$, according to the entries in the access control list of the participant defined by $\mathcal{A}$ (or $\emptyset$, if the participant is not authorized to access the ledger). This corresponds to "what the participant can see".

Recall that the database or a subset is a collection of keys and their values. We can simplify its representation by referring to the projection of the ledger l against the participant p (this is, the projection function $\mathcal{F}_\pi$ that is chosen projects the states of the virtual ledger that are accessible by p). The projection in the ledger $\mathcal{L}$ using the projection function $\mathcal{F}_p$ outputs a set of states $\{s_1, \ldots, s_n\}$ that the participant p can access, i.e., $d_{\mathcal{L},\mathcal{F}_p} = \{s_1, \ldots, s_n\}$. We use the notation $\overline{s}$ to represent the absence of a state in a projection. Consider the following projections, illustrated by Table 2:

- $d_{\mathcal{L},\mathcal{F}_{p_1}} = \{ s_1, \overline{s_2}, \overline{s_3} \} = s_1$
- $d_{\mathcal{L},\mathcal{F}_{p_2}} = \{ s_1, s_2, \overline{s_3} \} = s_1, s_2$
- $d_{\mathcal{L},\mathcal{F}_{p_3}} = \{ s_1, \overline{s_2}, s_3 \} = s_1, s_3$
- $d_{\mathcal{L},\mathcal{F}_{p_4}} = \{ \overline{s_1}, s_2, s_3 \} = s_2, s_3$
- $d_{\mathcal{L},\mathcal{F}_{p_5}} = \{ \overline{s_1}, s_2, \overline{s_3} \} = s_2$
- $d_{\mathcal{L},\mathcal{F}_{p_6}} = \{ \overline{s_1}, s_2, \overline{s_3} \} = s_2$

We can retrieve a projection $d_{\mathcal{L},\mathcal{F}_p}$ (or empty set) with the primitive `obtainVirtualLedger`. This primitive receives as input a ledger $\mathcal{L}$ and a projection function $\mathcal{F}_p$. Some projections are not unique (e.g., $d_{\mathcal{L},\mathcal{F}_{p_5}}$ and $d_{\mathcal{L},\mathcal{F}_{p_6}}$). The concept of projection is the basis for the DLT view, which we will define later in this section. Both

ledgers and virtual ledgers are abstractions to access a state, represented by a key-value store. Participants issue transactions to change the state. A transaction is defined as follows:

Definition 5 (Transaction). A transaction t is a tuple $(t_{id}, t, payload, \sigma_{K_s^P}(message), S_{tid}, target)$, where:

- t_{id} is a unique increasing sequence identifier for a transaction. This identifier allows one to construct a transaction ID, a unique identifier for a transaction. Transaction t_i precedes t_j, i.e., $t_i \preccurlyeq t_j$ if and only if $j > i$.
- t is the transaction timestamp
- *payload* is the transaction payload. The payload can carry arbitrary information (smart contract parameters, UTXO input value).
- a signature on the transaction $\sigma_{K_s^P}(message)$, where $message \doteq (t_{id}, t, target, payload)$.
- S_{tid}, a set of input states given as input to the transaction with transaction ID *sid*.
- *target* is the state ID to which the transaction refers.

A transaction takes as input a S_{tid} and outputs S'_{tid}. We define a primitive `VerifyTx(.)` that takes a set of initial states, a transaction, and a set of output states and outputs one if and only if the state transition is valid according to some algorithm ρ, i.e., `VerifyTx`$(S_{tid}, t_{id}, S'_{tid}, \rho = 1)$. Checking the validity of a transaction w.r.t. the states it changes implies re-running the transaction on its run environment. Transactions produce state changes. We define the state as:

Definition 6 (State). A state s is a tuple $(s_k, s_{k,v}, \mathcal{T}, \pi_k)$, where

- s_k is a unique identifier (the state's key).
- a transaction list $\mathcal{T}$, referring to that state, i.e., $\forall t \in \mathcal{T} : t.target = s_k$
- the value it holds $s_{k,v}$. The value of a state can be calculated using the set of transactions $\forall i, TS = \{t_i \subset \mathcal{T}, s \in \mathcal{S} : t_i.target = s_k\}$, i.e., the transactions referring to that state, in the following manner:

$$s_{k,v} = \begin{cases} \emptyset & i = 0 \\ apply(t_i, s_{k,i-1}) & i \leq |TS| \end{cases}$$

  where *apply* is a function that executes the payload of the transaction t_i on the state s_k. The data is the most recent state value, the result of the successive transformations (over the previous versions of the same state). The function *apply* is blockchain-dependent.
- a proof of state validity $\pi_k = \sigma_{K_s^P}(s_k, s_{k,v}, v)$, where $s_{k,v}$ is the value of state s_k at version v, and σ_m is the set of signatures of participants $P \subset \Upsilon$ that creates the proof, over a payload m.

A state has a unique reference (or key) s_k and a version v such that when v is updated, it yields $v' > v$. We denote the value pointed by that reference by $s_{k,v}$. If we omit the version, we refer to s_k as the latest value in a certain state. Thus, for all $k \neq k'$, s_k and s'_k represent the latest value of different states. In practice, the value of a state is the result of successively executing transactions over the same object. The value for $s_{k,v}$ or (s_n) can be calculated as follows, where transaction set $\{t_1, \ldots t_{k-1}\} \in TS$ are the transactions referring specifically to s_k:

$$s_{k,0} \xrightarrow{t_1} s_{k,1} \xrightarrow{t_2} \ldots \xrightarrow{t_{k-1}} s_{k,v}$$

Each ledger database stores states in its key-value store. The state identifier, s_k, is the key, while the tuple $(s_{k,v}, t, \pi_k)$ is the value. Each proof $\pi \in \Pi$ is a string accounting for the validity of the item it describes (e.g., signature over transaction).[4]

[4]In Bitcoin, for instance, the proof of validity for a transaction is the issuer's signature, along with a nonce whereby its hash begins with a certain number of zeroes and is smaller than a certain threshold (valid transaction within a valid block). In Hyperledger Fabric, the proof is a collection of signatures from the endorsing peer nodes that achieved consensus on the transaction's validity.

We define the *cardinality* of a state s_n as $|s_n|$ as the number of transactions that compose it. If s_n has a set of transactions $\mathcal{T} = \{t_1, \ldots, t_i\}$, then $|s_n| = i$. The state of a particular object can be reconstructed from the execution of all the transactions that refer to it. The global state is then the set of all states, the set $\mathcal{S}$. The set of states visible by a certain participant (states that authorize the participant to read/write/update) is a view of the blockchain.

Having introduced all the basilar concepts, we can define a DLT view:

Definition 7 (View). A view $\mathcal{V}$ is a projection of a virtual ledger $\mathcal{L}$, in the form of $(v_k, t_i, t_f, p, d_{\pi_{l,p}}, S, \Pi)$, where

— its key v_k, is a unique ID
— an initial time t_i and a final time t_f that restrict the states belonging to that view. A view may have no restriction on the temporal interval, i.e., all states that a participant p accesses through $d_{\pi_{l,p}}$ are included in the view.
— $p \subseteq \Upsilon$ is the set of participants associated with the view. A participant *upsilon* can be associated with one or more nodes, accessible by a blockchain access point ω.
— a projection function $d_{\pi_{\mathcal{L},p}}$ used to build the view.
— S corresponds to the set of versioned states that the participant in the view has access to (via the projection function).
— Π is a set of proofs accounting for the validity of a view (e.g., accumulator value for over states ordered by the last update).

The consolidated view, or the global view $\mathcal{V}$, is the set of all participant views, i.e., $\mathcal{V} = \cup_{i=0}^{i} p_i$, that captures the entire ledger $\mathcal{L}$.

Definition 8 (View Cardinality). Let there be a DLT view $v_{l,p}$ belonging to a participant p. A view $v_{l,p}$ has cardinality i when the number of states composing that view is i, i.e., $d_{\pi_{l,p}} = \{s_1, \ldots, s_i\}$. In other words, $|v_{l,p}| = i$.

Definition 9 (DLT Domain). t is a tuple $(d, s_k, s_{k,v}, \mathcal{F}_d, P, value)$, where:

— d is the identifier of the domain
— s_k is the state key to which that domain refers.
— $s_{k,v}$ is the state value corresponding to s_k.
— projection function that generates the state value $s_{k,v}$ of domain d indexed by s_k.
— P is the set of participants that can read the value v of state s_k on a domain d.

Domains represent private relationships; they capture how many participants can share the same state. Domains then capture if a state is accessible (value readable) or not by a set of participants. The same state's key can have different values on different domains (depending on the projection function generating the domain). Two participants sharing the same domain does not mean having the same view.

Definition 10 (View Equivalence). Let there be a ledger $\mathcal{L}$ composed of a set of views $\{v_1, v_2, \ldots, v_n\} \in \mathcal{V}$, holding respectively the sets of states $\{S_1, S_2, \ldots, S_n\} \in \mathcal{S}$, i.e., there are the pairs $\{(v_1, S_1), (v_2, S_2), \ldots (v_n, S_n)\}$. There is view equivalence, denoted by $\texttt{equivalent}(v_i, v_j)$ if, for any pair of views $v_i, v_j \in \mathbb{V}$ there is a bijection $\varphi : s_i \rightarrow s_j$ such that if both sets of states from the views are the same, their views are equivalent, i.e., $\varphi(s_i) = s_j \implies \forall s \in s_i, s$, could replace all $s' \in s_j \implies v_i \equiv v_j$.

Following the example of Table 2, v_1 and v_6 are equivalent views because the states that are accessible to those views are the same. However, those views are unequal, as the other parameters might change.

Definition 11 (View Transparency). Let there be a ledger $\mathcal{L}$ with a set of participants Υ and a set of DLT views $\{v_1, \ldots, v_n\} = \mathcal{V}$. We define the transparency grade κ of a DLT view v_v, denoted as $\kappa(v_v)$, as the ratio of

participants who can access the states encoded in that view. More formally,

$$\kappa(v_v) = \frac{\sum_{i \neq v}^{n} \forall v_i[\texttt{equivalent}(v_n, v_j) + 1]}{|\mathcal{V}|}$$

This concept is useful to understand how many participants can access a certain set of states. Taking the example from Table 2, $\kappa(v_{l,\mathcal{F}_{p_6}}) = \frac{2}{6}$ because the view created by projecting the ledger l with $d_{\mathcal{F}_{p_6}}$ is equivalent to v_5 (summing with the view being compared). Therefore, two out of six participants can access the same set of states (i.e., each participant has a different view, apart from p_5 and p_6).

4 BUNGEE, A MULTI-PURPOSE VIEW GENERATOR

In this section, we present BUNGEE. First, we present the system, key management processes, and adversary models. After that, we present the snapshot process. Next, we present how views are built and then merged. The section ends with discussion on the processes of creating snapshots and views, as well as merging views.

4.1 System Model

We consider an asynchronous distributed system, the DLT, that hosts a ledger $\mathcal{L}$. Three types of participants interact with the ledger: (i) participants Υ: entities that transact on the network (can use `read` and `write` operations) via the nodes that their AP exposes; (ii) nodes $\mathcal{N}$, who hold the full state of the DLT, and contribute to the consensus of the latter; and (iii) view generators $\mathbb{G}$, programs that build views, via a node that has access to the target participant of the view. This implies that a view generator trusts the node that is the access point to the DLT. Each DLT is assumed to be able to preserve its safety and liveness abilities despite the possible existence of malicious nodes. This implies that building and operating views based on networks that cannot guarantee safety properties (e.g., DLT forks due to attack) are invalid.

Key management. Each participant $p \in \Upsilon$, node $n \in \mathcal{N}$, and view generator $\mathbb{G}$ is identified by a pair of keys (K_p^p, K_k^p), (K_p^n, K_k^n), and $(K_p^{\mathbb{G}}, K_k^{\mathbb{G}})$, respectively. The private key is the signing key, while the public key is the verification key. The generated keys are independent of all other keys, implying that no adversary with limited computational resources can distinguish a key from one selected randomly. We assume that keys are generated and distributed in an authenticated channel, preserving integrity; digital signatures cannot be forged. We say that an entity x signs a message m with its private key with the following notation: $\texttt{sign}_x(m)$. Verifying a message m with the public key from x can be done with a `verify` primitive, which outputs one if the message was correctly signed by m, i.e., $\texttt{verify}(m, x, K_x^p) = 1$, and 0 otherwise.

4.2 Adversary Model

The DLT where view generators operate is trusted, meaning that most internal nodes are honest, and thus the network is trusted. Given this assumption, there can be different adversary models for nodes, generators, and view generators. Nodes can be honest by following the DLT protocol, establishing consensus with other honest nodes, and reporting the actual status of the DLT to participants who request it. Nodes can, instead, be malicious, i.e., Byzantine, being able to deviate from the protocol and falsely report the DLT status to participants (endangering the creation of truthful views). Nodes can be malicious but cautious, meaning that they are only malicious if there are no accountability checks that can penalize them (i.e., if they know that they cannot get caught).

View generators constitute a trusted group with the participant that is the target of the view because the generator needs the participant's credentials to access a (private) subset of the ledger. We then assume that each participant runs its view generator. View generators can only build views for participants whose keys they do not control if the ledger has no partition (i.e., it is public). Since participants might access DLT partitions from different nodes, the trust group (participant, view generator) does not include a node or set of nodes, i.e., view generators and DLT participants are independent of the nodes that sustain the DLT.

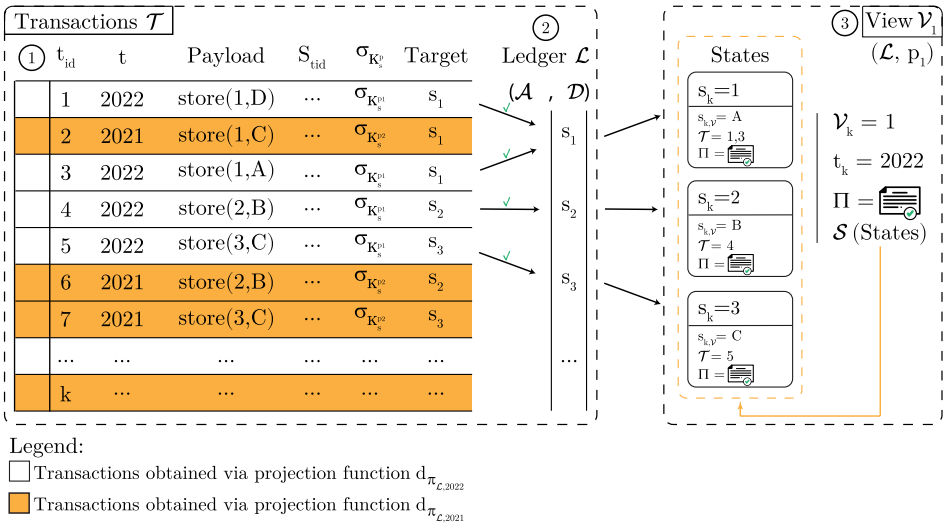

Fig. 3. View generation process for participant p_1, ledger $\mathcal{L}$, using projection functions $d_{\pi_{\mathcal{L},2021}}$ (yellow rows) and $d_{\pi_{\mathcal{L},2022}}$ (white rows).

4.3 View Generation Process Overview

BUNGEE constructs views from a set of states from an underlying DLT called a snapshot. This is done by obtaining a virtual ledger on behalf of a certain participant with a projection function. Then, each state accessible by the participant is collected in the snapshotting phase. The states are processed, and a representation of the ledger is built to which the participant has access. We can think of the snapshot as the capture of available transactions from the perspective of the participant in a table (*c.f* Figure 3, step (1)) upon having the necessary permissions from the ledger ((2)). Right after that, in the view building phase, a view is built from the virtual ledger that the view generator can access by temporarily limiting the states that one can see ((3)). Views can be stored in a local database, providing relational semantics and rich queries. Views are assured to provide provenance, i.e., BUNGEE can trace each component constituting a view down to the transaction.

After that, the view merging phase (optional) comprises merging views into an integrated one (see Section 4.7). For that, an extended state is created from the states present in each view that share the same key. Following that step, a merging algorithm is applied to the extended state. Finally, each view generator signs the integrated view, which can optionally be published in a public forum. The publication in a public forum can be decided by the participants that generate views (social consensus).

Let us focus on the high-level snapshot generation and view generation processes, as exemplified in Figure 3. In this example, we are building two views $\mathcal{V}_1$ and $\mathcal{V}_2$ (from participants p_1 and p_2, which capture states whose projection functions are $d_{\pi_{\mathcal{L},2022}}$ and $d_{\pi_{\mathcal{L},2021}}$, respectively. The semantics for the projection functions are simple: $d_{\pi_{\mathcal{L},2022}}$ refers to transactions timestamped as of 2022, and $d_{\pi_{\mathcal{L},2021}}$ refers to transactions timestamped as of 2021. In the figure, we focus on building $\mathcal{V}_1$ - the white rows. Thus, the transactions timestamped 2021 (in the yellow rows) are not captured on view $\mathcal{V}_1$. Transactions t_1, t_2 and t_3 alter state s_1, but t_2 belongs to view $\mathcal{V}_2$, so its not included. Transaction t_4 changes stores B as the value of s_3, while transaction t_5 sets s_3 as C.

Once the three steps are completed, BUNGEE returns the views (the generated and the integrated views) to the client application (for example, a blockchain migration application). For example, the client application might use BUNGEE to retrieve snapshots that refer to a period relevant to an audit. Due to BUNGEE's modularity, adding support for different applications is facilitated. Next, we present each phase depicted in this overview in finer detail.

ALGORITHM 1: Snapshotting of ledger $\mathcal{L}$ through node n and participant p, via projection function $\mathcal{F}_p$

Input: Access point AP, participant p, projection function $\mathcal{F}_p$, snapshot identifier $\texttt{snapshot}_{id}$
Output: Snapshot from participant p through node n, snapshot

1 $\texttt{snapshot.id} \leftarrow \texttt{snapshot}_{id}$
2 $\texttt{snapshot.v} \leftarrow 1$
3 $\texttt{snapshot.sb} \leftarrow \emptyset$
4 $\texttt{snapshot}.t_i \leftarrow \perp$
5 $\texttt{snapshot}.t_f \leftarrow \perp$
6 $t_{it} \leftarrow \infty$ ▷ temporary variable to hold minimum state timestamp to date
7 $t_{ft} \leftarrow 0$ ▷ temporary variable to hold maximum state timestamp to date
8 $n = \omega^{-1}(p)$ ▷ choose any available node
9 $\mathcal{L} = \texttt{obtainDLT}(n)$ ▷ depends on the DLT client implementation
10 $d_{\mathcal{L},\mathcal{F}_p} = \texttt{obtainVirtualLedger}(\mathcal{L}, \mathcal{F}_p)$ ▷ obtain projection of $\mathcal{L}$ according to p
11 **foreach** $s_k \in d_{\mathcal{L},\mathcal{F}_p}$ **do**
12 $s_{k,it} \leftarrow \emptyset$ ▷ the timestamp of the first transaction applied to state s_k
13 $s_{k,lt} \leftarrow \emptyset$ ▷ the timestamp of the last transaction applied to state s_k
14 $\texttt{snapshot}.sb[s_k].s_k = s_k$
15 $\texttt{snapshot}.sb[s_k].version = d_{\mathcal{L},\mathcal{F}_p}[s_k].T.length$
16 $\texttt{snapshot}.sb[s_k].\texttt{latestValue} = d_{\mathcal{L},\mathcal{F}_p}[s_k].s_{k,v}$
17 $\texttt{snapshot}.sb[s_k].T = d_{\mathcal{L},\mathcal{F}_p}[s_k].T$ ▷ save list of transactions referring to each state key
18 $s_{k,it} = d_{\mathcal{L},\mathcal{F}_p}[s_k].T[0]$ ▷ transaction list is ordered chronologically
19 $s_{k,lt} = d_{\mathcal{L},\mathcal{F}_p}[s_k].T.length$
20 **if** $s_{k,it} < t_{it}$ **then**
21 $t_{it} = s_{k,it}$ ▷ update the auxiliary first timestamp
22 **end if**
23 **if** $s_{k,lt} > t_{ft}$ **then**
24 $t_{ft} = s_{k,lt}$ ▷ update the auxiliary last timestamp
25 **end if**
26 **end foreach**
27 $\texttt{snapshot}.t_i = t_{it}$
28 $\texttt{snapshot}.t_f = t_{ft}$
29 **return** snapshot

4.4 Snapshot

A snapshot is a set of states that a certain stakeholder can access, plus proof of the validity of that state. We view each state as a versioned (key, value) store. A snapshot has a snapshot identifier id, a version v, a participant p, a set of states bins, sb, an initial time t_i that refers to the timestamp of the first transaction of any of the states belonging to sb, a final time t_f that refers to the timestamp of the last transaction of any of the states belonging to sb, i.e.,$\texttt{snapshot} \doteq \{id, v, sb, t_i, t_f\}$. Each state bin is indexed by a state id s_k, the latest value to that key, $s_{k,\overrightarrow{v}}$, a version v that refers to the number of transactions applied on the state key s_k to produce the latest value $s_{k,\overrightarrow{v}}$ and a list of transactions T referring to that state (as in Definition 7). Versioning snapshots allows one to efficiently build snapshots from older snapshots (i.e., building snapshots from incremental changes from older snapshots).

Algorithm 1 depicts the snapshotting process. The snapshot phase occurs when the BUNGEE client requests the beginning of the view integration process to a node n on behalf of the participant p (line 8). After that,

ALGORITHM 2: Constructing a view $\mathcal{V}$ of ledger $\mathcal{L}$ with snapshot snapshot, from the perspective of participant p.

Input: Snapshot snapshot, view id id, initial time t_i, final time t_f
Output: View $\mathcal{V}$

1 $\mathcal{V}.k \leftarrow id$
2 $\mathcal{V}.t_i \leftarrow t_i$
3 $\mathcal{V}.t_f \leftarrow t_f$
4 $\mathcal{V}.d_{\pi_{l,p}} \leftarrow \texttt{snapshot}.\mathcal{F}_p$
5 $\mathcal{V}.p \leftarrow \texttt{snapshot}.p$
6 $\mathcal{V}.\Pi \leftarrow \perp$
7 $\mathcal{V}.S_{k,v} \leftarrow \perp$
8 **if** $t_i < \texttt{snapshot}.t_f$ ***OR*** $t_f > \texttt{snapshot}.t_i$ **then**
9 | return; ▷ there are no intersecting states that we want to capture, on the snapshot
10 **end if**
11 ▷ each $sb = \{s_k, s_{k,\overrightarrow{v}}, \text{v}\}$
12 **foreach** $s_k \in \texttt{snapshot}.sb$ **do**
13 | **foreach** $t \in s_k$ **do**
14 | | **if** $t.timestamp < t_i$ ***OR*** $t.timestamp > t.f$ **then**
15 | | | $\texttt{snapshot}.sb[s_k] \leftarrow \texttt{snapshot}.sb[s_k].\mathcal{T} \setminus t$ ▷ removes transaction that is not within the specified time frame
16 | | **end if**
17 | **end foreach**
18 | $\mathcal{V}.S_{k,v} \leftarrow \texttt{snapshot}.sb[s_k]$
19 **end foreach**
20 $\mathcal{V}.\Pi \leftarrow sign_{\mathbb{G}}(\mathcal{V})$
21 **return** $\mathcal{V}$

the node connects to the DLT. Upon a successful connection, n retrieves the ledger (line 9). Obtaining a list of states from a ledger requires checking all transactions that performed state updates. For each transaction, a BUNGEE has to check its target. BUNGEE creates a new state if there is no state key with a target equal to the current transaction. The version of the new state is one. Then, BUNGEE runs the transaction's payload against the current state value (empty at initialization). Otherwise, if the transaction target refers to an existing state key, run the transaction payload against the state's current value, yielding the new value and incrementing the version by one. This process outputs a list of states. According to the participant's perspective, the process is abstracted by the ledger's projection (according to the participant's perspective) that the algorithm uses (line 11). The snapshot maps each state to a state bin. For each state, we collect its key (line 15), version (line 16), latest value (line 17), the auxiliary first timestamp (line 18), and auxiliary latest timestamp (line 19). After that, the first and last timestamps are updated (lines 28 and 29), and, at last, the algorithm returns a snapshot.

4.5 View Building

This section explains how views are built. A view generator can generate a set of views depending on the input p. The following steps occur for each view to be built: first, the view generator generates a snapshot. After that, the snapshot is limited to a time interval and signed by the view generator.

Algorithm 2 shows the process of building a view from a snapshot. First, the view generator temporarily limits each included state, proceeding to abort if no states are within its boundaries (line 8). If there are, each state in the snapshot is included if it belongs to the temporal limit (line 18) and removed otherwise (line 15). Finally, the view generator signs the view (line 20) and returns it to the client application (line 21).

ALGORITHM 3: Merging a set of views $\mathcal{V} = \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n$, where each view was built referring to participant $p_1, p_2, \ldots p_n$ respectively by a set of view generators $\mathbb{G} = \mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_n$

Input: Views to be merged $\mathcal{V} = \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_n$, merging algorithm $\mathcal{M}$
Output: Integrated view $\mathcal{I}$

1 $\mathcal{S} \leftarrow []$ ▷ state list $\mathcal{S}_{\mathcal{V}_1,\ldots,\mathcal{V}_n}$ ($\mathcal{S}$ for simplicity) where each index (representing a state key) maps to tuple of values from referring to that key, from each view to be merged
2 $\mathcal{I}.t_i \leftarrow \emptyset$
3 $\mathcal{I}.t_f \leftarrow \emptyset$
4 $\mathcal{I}.d_{\pi_{l,p}} \leftarrow \bigcup_{i=0}^{n} \mathcal{V}_n.d_{\pi_{l,p}}$
5 $\mathcal{I}.p \leftarrow \bigcup_{i=0}^{n} \mathcal{V}_n.p$
6 $\mathcal{I}.\Pi \leftarrow \perp$
7 $\mathcal{I}.S_{k,v} \leftarrow \perp$
8 **foreach** $v \in \mathcal{V}$ **do**
9 **foreach** $s \in v.S_{k,v}$ **do**
10 **if** $s \in \mathcal{S}$ **then**
11 $\mathcal{S}[\overrightarrow{s.k}] = \mathcal{S}[\overrightarrow{s.k}] \cup \overrightarrow{s_{k,v}}$ ▷ if state exists, add value referring to that state, from current view
12 $\mathcal{S}[\overrightarrow{s.k}].version \leftarrow \mathcal{S}[\overrightarrow{s.k}].version + 1$
13 **end if**
14 **else**
15 $\mathcal{S}[\overrightarrow{s.k}] = \overrightarrow{s_{k,v}}$ ▷ otherwise, initialize state key list
16 $\mathcal{S}[\overrightarrow{s.k}].version \leftarrow 0$
17 $\mathcal{S}[\overrightarrow{s.k}].metadata \leftarrow \{MERGE - INIT\}$
18 **end if**
19 **end foreach**
20 **end foreach**
21 $\mathcal{I}.S_{k,v} = \texttt{call}_{\texttt{algorithm}\mathcal{M}}(\mathcal{S})$ ▷ OPTIONAL. Computes the state list of the integrated view according to $\mathcal{M}$ (see for example algorithm 4)
22 $\mathcal{I}.d_{\pi_{l,p}} \leftarrow \mathcal{I}.d_{\pi_{l,p}} \cup \{\mathcal{M}\}$ ▷ add reference to the merging algorithm
23 $\mathcal{I}.t_i = \texttt{min}\{\mathcal{I}.S_{k,v}.t_i\}$ ▷ initial timestamp correspond to the initial timestamp of the processed states
24 $\mathcal{I}.t_f = \texttt{min}\{\mathcal{I}.S_{k,v}.t_f\}$
25 $\mathcal{I}.\Pi \leftarrow sign_{\mathbb{G}}(\mathcal{I})$ ▷ signed collectively by $\mathbb{G}$
26 **return** $\mathcal{I}$

4.6 Merging Views

In this section, we describe how to merge views in a privacy-preserving way. The merging of views creates an integrated view $\mathbb{I}$ from a set $\mathcal{V}$ of input views. The idea is to compare the state keys indexed by every view and their value according to a merging algorithm $\mathcal{M}$ that is given as input. This merging algorithm controls how the merge is performed, and therefore, a user can set up policies that comply with the privacy needs (of all participants).

Algorithm 3 shows the procedure for merging views. The algorithm receives the views to be merged and returns an integrated (or consolidated) view as input. We initialize an auxiliary list $\mathcal{S}_{\mathcal{V}_1,\ldots,\mathcal{V}_n}$ (on line 1) that holds all the values (coming from different views) for each state key. We propose a construct called an extended state. An extended state is a state where each state key maps to a set of values. Additionally, an extended state has a *metadata* field holding a list of operations applied to that extended state.

Definition 12. An *Extended State* $\overrightarrow{s}$ is a tuple $\overrightarrow{s_k}, \overrightarrow{s_{k,v}}, t, \pi_k, metadata, version$), where

— $\overrightarrow{s_k}$ is a unique identifier (the state's key);
— $\overrightarrow{s_{k,v}}$ is a list of values;
— a transaction list $\mathcal{T}$;
— a proof of state validity π_k;
— metadata, which holds a list of operations that have been applied to the extended state;
— version, a monotonically increasing integer. The counter increases when an update is done to the extended state (the number of elements in the metadata field is the same as the version).

Thus, each index of the set of extended states $\mathcal{S}$ will index all different values for each key for all the views to be merged, i.e.,

$$\mathcal{S}_{\mathcal{V}_1,\ldots,\mathcal{V}_n} = \left\{\forall s_i \in \mathcal{S} : \exists k_i \in s_i : k_i \implies \left(s_{\mathcal{V}_{1(k_i,v)}}, \ldots, s_{\mathcal{V}_{n(k_i,v)}}\right)\right\}$$

After we initialize the list of extended states, in Algorithm 3, we initialize the integrated view properties: its initial timestamp (line 2), final timestamp (line 3), projection functions (taken as the union of the projection functions of all the views, on line 4), participants (the participants from each view, on line 5), a set of proofs (line 6) and a set of states (line 7). The set of states to be assigned as the set of states of the integrated view is a function of the processed auxiliary set of states $\mathcal{S}$. After all, we check each state key to merge each view. If the tested state is already on the auxiliary state set (line 10), then we add its value $\overrightarrow{s_{k,v}}$ as a value for the current extended state key (line 11). This outputs a list of values (between one and the number of views to be merged) for each extended state key. Otherwise, we set a new extended state, adding the current state value (as the first value for that key, on line 15).

On line 21, we apply an optional view processing phase by giving our list of states $\mathcal{S}$ to an arbitrary algorithm that needs to respect a simple interface and functionality (later defined). After that, we add algorithm $\mathcal{M}$ as a projection function for $\mathcal{I}$ for future traceability and auditing. Next, we adjust the initial and final timestamps (lines 23 and 24) because the merging algorithm might have changed the time boundaries of the included states (for example, the state corresponding to the lowest timestamp might have been removed). All view generators must sign $\mathcal{I}$ (line 25) to promote accountability. Signing the integrated view can be distributed using a multi-signature algorithm (for example, BLS Multi-Signatures [52]).

Each merging phase has an optional application of a merging algorithm $\mathcal{M}$, which dictates how the merge is carried out (otherwise, all states are included without further processing). We define a simple interface for merging algorithms: a merging algorithm receives a set of extended states as input and outputs a set of extended states. The functionality of the merging functions should be: (1) apply arbitrary operations on the set of extended states, (2) add a reference to the current merging algorithm to the *metadata* field of each extended state key that is altered, (3) increase the version of each extended state key that is altered. Each merging algorithm should be public and well-known to the parties involved.

Examples of merging algorithms are:

— *Pruning:* removes the values coming from a particular view.
Algorithm 4 prunes the values belonging to a particular view from a set of extended states. Note that times do *not* need to be updated because they are recalculated in steps 23 and 24 of Algorithm 3. Applications include removing sensitive information in the context of existing regulations and laws.

4.7 Example: Merging Two Views

In this section, we graphically show an example of a merge view, by applying Algorithm 3 (merge view) and `MERGE-ALL`. Informally, `MERGE-ALL` works by keeping the values from both views included in the final view (a rather simple merge algorithm). Let us consider two views $\mathcal{V}_1$ and $\mathcal{V}_2$ (create from the table of Figure 3), and its

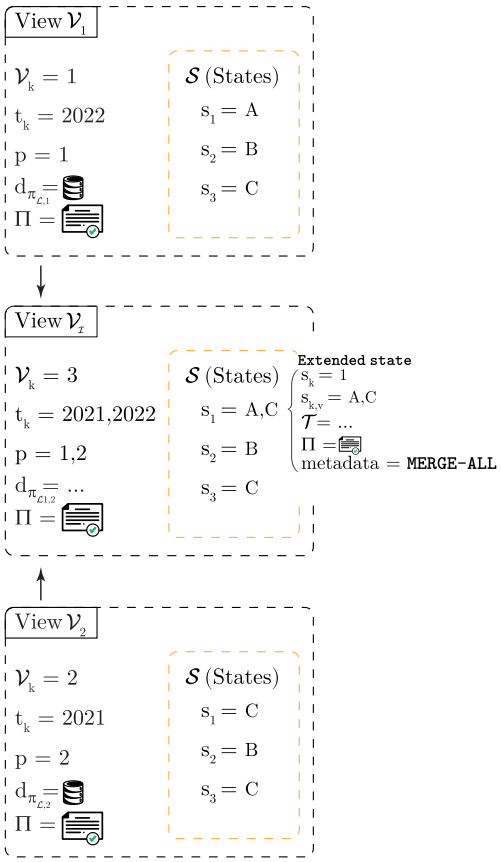

Fig. 4. Merging of views $\mathcal{V}_1$ and $\mathcal{V}_2$ into a consolidated view $\mathcal{V}_I$ according to merging algorithm MERGE-ALL.

ALGORITHM 4: Merging algorithm example – PRUNE (by $\mathcal{V}_1$)

Input: The set of states to be processed $\mathcal{S}$
Output: A processed set of states $\mathcal{S}'$

$\mathcal{S}' \leftarrow \emptyset$ ▷
foreach $s \in \mathcal{S}$ **do**
  if $s_k[0]$ **then**
    $\mathcal{S}'[s_k] = \mathcal{S}'[s_k] \setminus s[0]$ ▷ if there exists a value for view $\mathcal{V}_1$, then remove that value from the state list
    $\mathcal{S}'[s_k].metadata \leftarrow$ PRUNE-VIEW-1
    $\mathcal{S}'[s_k].version \leftarrow \mathcal{S}'[s_k].version + 1$
  end if
end foreach
return $\mathcal{S}'$

merging into a consolidated view $\mathcal{V}_I$, c.f. Figure 4. View $\mathcal{V}_1$ and $\mathcal{V}_2$ differ on the value for s_1, A and C, respectively. The integrated view will hold an extended state with (1) a timestamp including both views, (2) references to the participants generating each view, (3) the joint projection function, (4) a set of proofs, and (5) a set of extended states. For s_1, we have included the different values from the different views.

5 DISCUSSION

In this section, we discuss BUNGEE. The proliferation of blockchain interoperability solutions is increasing interest in exploring cross-chain logic and the need to model and analyze it [53]. Our proposal constitutes the foundation to make sense of that diversity by allowing us to systematically create and integrate views from different blockchains. In this section, we discuss the studied research questions, with considerations on the integrity, accountability, and privacy of views.

5.1 RQ 1: Providing a Data Format for Views

Views can be generated from different sources, as long as they are accompanied by a valid proof. The existence of proofs on states is a proof of creation by the entities that created or executed the transactions referring to that state. For example, a signed transaction hash qualifies as proof of a transaction that makes part of the state proof (as many proofs as signed transactions referring to a certain state). On the other hand, views are also signed by the view generator that either generates, merges views, applies a merging algorithm or notarizes the view as true. This set of proofs allows independent parties to validate the truthfulness of the view (by verifying each state) and hold view generators accountable. In a permissioned environment, an auditor can confirm that the views are valid and complete. The metadata fields on each extended state and the views allow one to understand who, when, and how a view generator changes a certain view. However, more accountability measures can be implemented. In particular, if a view is only shared across the view generators that endorsed it, there might be limited exposure and, therefore, limited transparency. To enhance transparency, our key insight is to store a view in a public forum such as the InterPlanetary File System [54] (a distributed peer-to-peer file system maintained by a network of public nodes) or a public blockchain, similar to some related work [42, 43]. If a view is deemed false, automatic view conflict detection and resolution can occur.

For the network to enforce the integrity of views, we need two conditions to hold. First, each group of nodes that accesses a subset of a ledger (and thus creates a view) must have at least two elements. Second, there is at least one honest element for every group. Thus, an honest view generator connected to an honest node holds the knowledge of the view v, and publishes it. If a malicious node broadcasts a false view v', an honest node can dispute it. Disputes can be calculated by calculating the difference between views and checking the proofs constituting each view. In particular, if an instance of BUNGEE, on behalf of participant A, holds the knowledge of a pair of different views v, v' referring to the same participant at the same time frame, then one of the views is false. Thus, the creator of one of the views is malicious. An honest view generator can reconstruct the disputed view and compare it to the view publicized by the malicious participant.

It is unlikely that all participants are colluding to change the perception of the inner state because, in principle, participants have different interests; however, there might be several situations in which the whole network gains if it colludes (i.e., blockchain with financial information). The ledger is unreliable if all internal nodes collude because the safety properties cannot be guaranteed. We hypothesize that using a view similarity metric could be a good tool to assess the quality of the view merging process. In other words, one could systematically compare how the final integrated view is different from each view that composes it.

5.2 RQ 2: Privacy-preserving Integrated Views

In this paper, we have introduced how to create, merge, and process views. However, a challenge remains unsolved: how to share views in a decentralized way? How does one manage the lifecycle of a view, including its creation, endorsement, and dispute? Although the work of Abebe et al. [43] sheds some light on this, how can one verify that a view is false? The solution offered by Abebe et al. includes parties voting on an invalid view, but this does not solve the problem per se because if the source blockchain is private, there is no canonical answer. Suppose that at least one view from the integrated view comes from a private blockchain; the signatures of the view guarantee that a certain participant has voted on the validity of that view. This could introduce problems if

all participants collude to show a false view. However, assuming that at least one view generator is honest, the view generator could initiate a dispute with the suspect of a false view.

A view generator could use fraud proofs [55] to create disputes about the validity of views, allowing an efficient and decentralized view management protocol. Application clients can then use the proof field from views, states, and transactions to validate a certain fact on a ledger. However, when BUNGEE merges views, completeness may not be guaranteed because the merged view depends on each input view, and processing might be applied (including pruning), possibly leading to information being excluded. A case to apply pruning might be when sensitive data is recorded in a ledger and later removed from the processing stage or even to remove "obsolete" data from the blockchain and therefore contribute to efficient bootstrapping of light clients [56]. An interesting detail is that each view only includes the state and respective proofs in timeframe t_k. However, to ensure that it is possible to validate the view, a pointer to the validity of the latest state before t_k should be available.

Our integration process follows a semantic approach to information based on a conceptual standard data model that we define as a view. Thus, for each practical implementation of BUNGEE, there needs to be a mapping between the data model of the underlying blockchain and the view concept. All views being uniform, we can not only represent data in all blockchains, but we can merge views belonging to different blockchains. The applicability is to build a complete picture of the activity of a participant in each network, but it can also be used to disclose information according to an access control policy [57]. While selective access control to views has been explored, there is space to explore decentralized identity access control mechanisms to provide fine-grain access over views, leveraging the need to unify the different notions of identity that emerge from different blockchains.

The reader might inquire how BUNGEE would ensure the privacy that partial consistent blockchains attempt to enforce when views are unified and then shared. To address this problem, we envision two solutions: first, merging views requires *tacit* consent from all parties sharing the input views. If there is sensitive data, the data can be removed before the view is created, or later removed in the snapshotting phase. This is essentially encoded by the projection function $\mathcal{F}_p$ used to obtain the virtual ledger. The second solution is to encrypt the data (or hash the data) [57], so the resulting view contains obfuscated information or a notarization proof [58], respectively. However, the scientific community agrees that storing sensitive data on-chain, even if encrypted, is a bad security practice due to the threat of cryptographic algorithms being broken in the future [59–62]. Zero-knowledge proofs can also be explored as a vehicle to prove facts on a ledger by disclosing limited information about such facts [63]. We leave those interesting research paths for future work.

5.3 Considerations on Privacy

Privacy is of the utmost importance when dealing with views. There is intra-group privacy (within participants sharing the same domain, and view equivalence) and intra-blockchain privacy (where two disjoint groups of participants might have different view cardinality and view transparency). Likewise, we can consider the privacy of a merged view against the environment (participants from other systems that are interested in reading the created views).

Consider two groups of participants, α and β that share a disjoint set of domains d_α and d_β within a blockchain. These participant groups will at least not have access to the states a and b, for the first and second groups, respectively. The higher the view transparency within a domain, the more shared states exist, and therefore, the easier a merge operation becomes (namely because if everyone knows the states within a view, there is no need to run preprocessing over those states - only when sharing with the exterior). The states group α cannot access from group β are then given by $c = d_\alpha \setminus d_\beta$. Upon a merge, states $d_\alpha \cap d_\beta$ can be freely processed without intra-blockchain privacy concerns (i.e., both groups have access to those states) - so it is a matter of the appointed view generator to apply a commonly-agreed algorithm over those states, for public exposure.

Regarding sharing the unified view with the exterior, the consortium agrees on a common procedure to enhance privacy (e.g., pruning sensitive information that only should be shared within the consortium).

5.4 Future Work

There is ongoing work on the implementation of BUNGEE in a flagship interoperability project called Hyperledger Cacti, in the scope of standardizing asset transfers, within the IETF.[5] We would also like to empirically validate our work by providing an implementation of BUNGEE that can provide support for building blockchain migrator applications.

Another future work venue deals with the evolution of blockchains, and consequently the evolution of views. Our solution diverges from the classical definition of a view in databases [2] since the data shown is not up-to-date, nor easily updatable. We propose designing algorithms to update a view, perhaps similarly to how Git manages updates to versioned files [64]. An API that inspects and creates updates to views can be used by applications to efficiently use up-to-date data. Finally, we find potential in studying zero-knowledge proofs to enhance view privacy.

6 RELATED WORK

In this section, we present the related work.

Partial Consistency. Graf et al. propose the concept of partial consistency [4]. An implementation of this principle is given by blockchains with state partitions such as channels. Although blockchains that provide this property have existed for several years, e.g., Quorum [31], IOTA [32], Corda [30], Hyperledger Fabric, Hyperledger Besu [65], and Ripple [66], to the best of our knowledge, this is the first formalization of the concept. Some solutions build partial consistency realizations on top of blockchains [67], such Canton [33], but are not formalized, and thus privacy properties of such channels are not clear. Our concept of view brings another way to reason about blockchain sharding [68], where each validator that is part of a shard runs a view generator and can communicate the state of the shard to different blockchains. Sharding is a technique to improve throughput, typically in public blockchains. A sharding scheme offloads the transaction processing to several groups of nodes called shards [68]. A shard is thus a de-facto logical view that guarantees the integrity and correctness of states regarding the participants that can access those states. Like a shard, a view is a logical separation of the ledger according to each participant.

Decentralized privacy-preserving computation. Some work surveys [69, 70] privacy-preserving techniques for blockchain interoperability, which we emphasize the latest survey [25], that supports the need for blockchain views. Several surveys claim that most implementations focus on privacy-preserving techniques for permissionless homogeneous blockchains, while we focus on both permissionless and permissioned heterogeneous blockchains.

Indirectly related work includes the work from Kosba et al. [71], who propose a privacy-preserving decentralized smart contract system. The Enigma network [88] (uses multi-party computation technology to enforce smart contract privacy). Others use trusted hardware to enforce privacy [72], or zero-knowledge proofs [73, 74]. Contrarily to these works, BUNGEE focuses on delivering privacy for merged views, while the related work focuses on ensuring privacy in a single domain.

Generating views. Katsis and Papakonstantinou [75] has summarized view-based data integration techniques. Our approach follows a Global and Local as View [76] because views are created from a subset of the global state, but then can be merged and processed. We call the reader's attention to the survey on view integration techniques, mostly used in the database and business process management research areas [8]. Abebe et al. [43] have proposed the concept of external view, a construct to prove the internal state of permissioned blockchains. However, this concept only applies to private blockchains. We extend and generalize the concept of view so that

[5] https://datatracker.ietf.org/group/satp/about/

it can be used for both public and private blockchains, and thus for heterogeneous interoperability purposes (by allowing the integration and merging of views).

Some proposals in industry and academia propose general data models for cross-chain interaction, namely the Rosetta API, Quant Overledger's gateways [36, 77], Blockdaemon's Ubiquity API [78], Polkadot's XCMP [79, 80], and Cosmos's IBC [81]. The Rosetta API and Blockdaemon's Ubiquity API only support public blockchains. Quant Overledger supports public and private blockchains but does not allow them to realize complex operations such as merging views. Polkadot and Cosmos have the previous limitation and can only support blockchains created with Substrate and Tendermint, respectively. None of those are well-accepted standards. On the other hand, BUNGEE aims to create views independent of the underlying blockchains, aiming to follow the efforts of IETF's standardization group SATP [41, 42, 82].

View applications. In [57], views provided fine-grain dynamic access control over private data in Hyperledger Fabric. In addition to the applications referred to in Section 1, we identify some studies using the concept of view for different purposes. Some authors use views to perform audits of participants on different blockchains [83, 84]. In particular, a view is created and then merged with other views from the same participant on different blockchains to create a global view of the participant's activity. Applications are, for instance, cross-chain tax audit [85], or cross-chain portfolio tracking [86], and even cross-chain security, by representing and monitoring cross-chain state [87], all applications that could benefit from a more formal treatment that BUNGEE can provide.

7 CONCLUSION

Views directly support blockchain interoperability since it is easier to share the perspectives of all participants across heterogeneous DLTs. This enables complex orchestration of cross-blockchain services and supports the new research areas of DLT interoperability, including blockchain gateway-based interoperability. In this paper, we introduce the concept of blockchain view, a foundational concept for handling cross-chain state. Views represent different perspectives of blockchain participants, allowing one to reason about their different incentives and goals. We present BUNGEE, a system that can create views from a set of states according to a projection function, yielding a collection of states accessible by a certain participant. BUNGEE can create a snapshot by retrieving the state of a blockchain, and based on participants' permissions, build a view of the global state. After that, BUNGEE creates extended states, the basis for merging blockchain views. Different views (possibly from different blockchains) can be merged into a consolidated view, enabling applications such as cross-chain audits and analytics. Finally, we discuss different aspects of BUNGEE, including decentralization, security, privacy, and its applications. An important area for future work is the use of zero-knowledge proofs to enhance view privacy.

ACKNOWLEDGMENTS

We warmly thank Luis Pedrosa, the colleagues of the academic paper workforce at Hyperledger, and the colleagues of IETF's SATP working group for many fruitful discussions. We thank Afonso Marques, Iulia Mihaiu, Benedikt Putz, and Diogo Vaz for reviewing and providing valuable feedback on an earlier version of this paper. We thank the anonymous reviewers for suggestions and feedback that helped improve the paper. The work of Limaris Torres was done while she was with Blockdaemon.

REFERENCES

[1] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2017. The Bitcoin backbone protocol with chains of variable difficulty. In *Advances in Cryptology–CRYPTO 2017*, Jonathan Katz and Hovav Shacham (Eds.). Springer International Publishing, Cham, 291–323.

[2] Rafael Belchior, Sérgio Guerreiro, André Vasconcelos, and Miguel Correia. 2022. A survey on business process view integration: Past, present and future applications to blockchain. *Business Process Management Journal* ahead-of-print, ahead-of-print (Jan. 2022). DOI: http://dx.doi.org/10.1108/BPMJ-11-2020-0529

[3] Remco Dijkman. 2008. Diagnosing differences between business process models. In *International Conference on Business Process Management*. DOI: http://dx.doi.org/10.1007/978-3-540-85758-7_20

[4] Mike Graf, Daniel Rausch, Viktoria Ronge, Christoph Egger, Ralf Küsters, and Dominique Schröder. 2021. A security framework for distributed ledgers. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*. Association for Computing Machinery, 1043–1064. DOI: http://dx.doi.org/10.1145/3460120.3485362 event-place: New York, NY, USA.

[5] Mike Graf, Ralf Küsters, and Daniel Rausch. 2020. Accountability in a permissioned blockchain: Formal analysis of Hyperledger Fabric. (2020). DOI: http://dx.doi.org/10.1109/EuroSP48549.2020.00023

[6] Rafael Belchior, Luke Riley, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2022. Do you need a distributed ledger technology interoperability solution? *Distributed Ledger Technologies: Research and Practice* (Sept. 2022). DOI: http://dx.doi.org/10.1145/3564532 Just Accepted.

[7] Rafael Belchior, Jan Süßenguth, Qi Feng, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2023. A brief history of blockchain interoperability. (9 2023). DOI: http://dx.doi.org/10.36227/techrxiv.23418677.v3

[8] Rafael Belchior, S'ergio Guerreiro, Andr'e Vasconcelos, and Miguel Correia. 2022. A survey on business process view integration: past, present and future applications to blockchain. *Business Process Management Journal* 28, 3 (2022), 713–739.

[9] Ankur Lohachab, Saurabh Garg, Byeong Kang, Muhammad Bilal Amin, Junmin Lee, Shiping Chen, and Xiwei Xu. 2021. Towards interconnected blockchains: A comprehensive review of the role of interoperability among disparate blockchains. *ACM Comput. Surv.* 54, 7 (July 2021). DOI: http://dx.doi.org/10.1145/3460287 Place: New York, NY, USA Publisher: Association for Computing Machinery.

[10] 2022. FTX: An Overview of the Exchange and Its Collapse. (2022). https://www.investopedia.com/ftx-exchange-5200842

[11] 2023. Centralised Exchanges are Terrible at Holding Your Money: A Timeline of Catastrophes - LocalCryptos Blog. (2023). https://blog.localcryptos.com/centralised-exchanges-are-terrible-at-holding-your-money/

[12] 2022. Crypto Hack: The Mt. Gox Tragedy | CoinMarketCap. (2022). https://coinmarketcap.com/alexandria/article/crypto-hack-the-mt-gox-tragedy

[13] Rafael Belchior, Andr'e Vasconcelos, S'ergio Guerreiro, and Miguel Correia. 2021. A survey on blockchain interoperability: Past, present, and future trends. *ACM Comput. Surv.* 54, 8 (October 2021). DOI: https://doi.org/10.1145/3471140

[14] Rafael Belchior, Peter Somogyvari, Jonas Pfannschmidt, André Vasconcelos, and Miguel Correia. 2023. Hephaestus: Modeling, analysis, and performance evaluation of cross-chain transactions. *IEEE Transactions on Reliability* (2023), 1–15. DOI: https://doi.org/10.1109/TR.2023.3336246

[15] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. 2022. zkBridge: Trustless Cross-chain Bridges Made Practical. (2022). DOI: http://dx.doi.org/10.48550/arXiv.2210.00264 arXiv:cs/2210.00264

[16] Simeon Armenchev, Rafael Belchior, Dimo Dimov, Emil Ivanichkov, Zahary Karadjov, Kristin Kirkov, Petar Kirov, and Yordan Miladinov. 2023. DendrETH: A smart contract implementation of the Ethereum light client sync protocol. (2023). https://github.com/metacraft-labs/DendrETH. Accessed: 21-June-2023.

[17] Thomas Chen, Hui Lu, Teeramet Kunpittaya, and Alan Luo. 2022. A Review of Zk-SNARKs. (2022). DOI: http://dx.doi.org/10.48550/arXiv.2202.06877 arXiv:cs/2202.06877

[18] Alex Biryukov and Sergei Tikhomirov. 2019. Security and privacy of mobile wallet users in Bitcoin, Dash, Monero, and Zcash. 59 (2019), 101030. DOI: http://dx.doi.org/10.1016/j.pmcj.2019.101030

[19] Thomas Hardjono, Alexander Lipton, and Alex Pentland. 2019. Toward an interoperability architecture for blockchain autonomous systems. 67, 4 (2019), 1298–1309. DOI: http://dx.doi.org/10.1109/TEM.2019.2920154

[20] Rafael Belchior, Luke Riley, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2023. Do you need a distributed ledger technology interoperability solution? *Distrib. Ledger Technol.* 2, 1 (2023), 37 Pages. DOI: http://dx.doi.org/10.1145/3564532

[21] Rafael Belchior, Andr'e Vasconcelos, Miguel Correia, and Thomas Hardjono. 2022. Hermes: Fault-tolerant middleware for blockchain interoperability. *Future Generation Computer Systems* 129, (2022), 236–251.

[22] Rafael Belchior, Luke Riley, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2022. Do You Need a Distributed Ledger Technology Interoperability Solution? (2022). DOI: http://dx.doi.org/10.36227/techrxiv.18786527.v1

[23] Rafael Belchior, Jan Süßenguth, Qi Feng, Thomas Hardjono, André Vasconcelos, and Miguel Correia. 2023. A brief history of blockchain interoperability. (Sept. 2023). DOI: http://dx.doi.org/10.36227/techrxiv.23418677.v3 Citation Key: BelchiorBriefHistoryBlockchain2023a.

[24] EU Blockchain Observatory Forum. 2023. The Current State of Interoperability Between Blockchain Networks. EU Blockchain Observatory Forum Note. (Nov. 2023). https://www.eublockchainforum.eu/sites/default/files/reports/EUBOF_Interoperability%20Report-30112023.pdf [Online]. Available: https://www.eublockchainforum.eu/sites/default/files/reports/EUBOF_Interoperability%20Report-30112023.pdf

[25] Andre Augusto, Rafael Belchior, Miguel Correia, Andre Vasconcelos, Luyao Zhang, and Thomas Hardjono. 2023. SoK: Security and privacy of blockchain interoperability. http://tinyurl.com/sok-sp-interop Citation Key: SoKSecurityPrivacy.

[26] Terje Haugum, Bjørnar Hoff, Mohammed Alsadi, and Jingyue Li. 2022. Security and privacy challenges in blockchain interoperability - A multivocal literature review. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE'22)*. Association for Computing Machinery, New York, NY, USA, 347–56.

[27] Ruoyu Yin, Zheng Yan, Xueqin Liang, Haomeng Xie, and Zhiguo Wan. 2023. A survey on privacy preservation techniques for blockchain interoperability. *Journal of Systems Architecture* (Apr. 2023), 102892. DOI: http://dx.doi.org/10.1016/j.sysarc.2023.102892

[28] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger Fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys'18)*. Association for Computing Machinery. DOI: http://dx.doi.org/10.1145/3190508.3190538 event-place: New York, NY, USA.
[29] Hyperledger. 2022. Private data Hyperledger-Fabricdocs master documentation. (2022). https://hyperledger-fabric.readthedocs.io/en/release-2.2/private-data/private-data.html
[30] R3 Foundation. 2021. R3's Corda Documentation. (2021). https://docs.r3.com/
[31] JP Morgan. 2017. Quorum White Paper. (2017). https://github.com/jpmorganchase/quorum/blob/master/docs/QuorumWhitepaperv0.2.pdf
[32] Wellington Fernandes Silvano and Roderval Marcelino. 2020. Iota Tangle: A cryptocurrency to communicate Internet-of-Things data. *Future Generation Computer Systems* 112 (Nov. 2020), 307–319. DOI: http://dx.doi.org/10.1016/j.future.2020.05.047
[33] Canton's team. 2021. Introduction to Canton Daml SDK 2.1.1 documentation. (2021). https://docs.daml.com/canton/about.html
[34] Peter Wegner. 1996. Interoperability. *ACM Computing Surveys (CSUR)* 28, 1 (1996), 285–287. Publisher: ACM New York, NY, USA.
[35] EY. 2022. EY announces general availability of EY Blockchain Analyzer: Reconciler. (2022). https://www.prnewswire.com/news-releases/ey-announces-general-availability-of-ey-blockchain-analyzer-reconciler-301544701.html
[36] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. 2021. A survey on blockchain interoperability: Past, present, and future trends. *Comput. Surveys* 54, 8 (May 2021), 1–41. http://arxiv.org/abs/2005.14282
[37] HMN Dilum Bandara, Xiwei Xu, and Ingo Weber. 2019. Patterns for blockchain data migration. (June 2019). http://arxiv.org/abs/1906.00239
[38] Martin Westerkamp and Axel Küpper. SmartSync: Cross-blockchain smart contract interaction and synchronization. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (2022-05). 1–9. DOI: http://dx.doi.org/10.1109/ICBC54727.2022.9805524
[39] Catarina Pedreira, Rafael Belchior, Miguel Matos, and André Vasconcelos. 2022. Securing Cross-Chain Asset Transfers on Permissioned Blockchains. (June 2022). DOI: http://dx.doi.org/10.36227/techrxiv.19651248.v3
[40] André Augusto, Rafael Belchior, André Vasconcelos, and Thomas Hardjono. 2022. Resilient Gateway-Based N-N Cross-Chain Asset Transfers. (Nov. 2022). DOI: http://dx.doi.org/10.36227/techrxiv.20016815.v2
[41] SAT working group. 2022. Re: [Sat] SAT entity identifiers and DIDs. (2022). https://mailarchive.ietf.org/arch/msg/sat/4WOAW1JEFRBU6TQHU1PNgdob6cs/
[42] Rafael Belchior, André Vasconcelos, Miguel Correia, and Thomas Hardjono. 2021. HERMES: Fault-tolerant middleware for blockchain interoperability. *Future Generation Computer Systems* (March 2021). DOI: http://dx.doi.org/10.36227/TECHRXIV.14120291.V1
[43] Ermyas Abebe, Yining Hu, Allison Irvin, Dileban Karunamoorthy, Vinayaka Pandit, Venkatraman Ramakrishna, and Jiangshan Yu. 2021. Verifiable observation of permissioned ledgers. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.
[44] Inside Bitcoins. 2022. What is Terra LUNA - Explaining the LUNA Crash. (May 2022). https://insidebitcoins.com/news/explaining-the-luna-crash
[45] Hyperledger Foundation. 2020. Hyperledger Fabric Private Data. (2020). https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html
[46] Nadia Hewett, Wolfgang Lehmacher, and Yingli Wang. 2019. Inclusive deployment of blockchain for supply chains. *World Economic Forum*.
[47] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. 2019. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research* 57, 7 (April 2019), 2117–2135. DOI: http://dx.doi.org/10.1080/00207543.2018.1533261
[48] Massimo Bartoletti, Stefano Lande, Livio Pompianu, and Andrea Bracciali. 2017. A general framework for blockchain analytics. In *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. 1–6.
[49] Lars Brünjes and Murdoch J. Gabbay. 2020. UTxO-vs account-based smart contract blockchain programming paradigms. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 73–88.
[50] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukoli.c, Sharon Weed Cocco, and Jason Yellick. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys'18)*, Association for Computing Machinery, Porto, Portugal, 1–15. DOI: https://doi.org/10.1145/3190508.3190538
[51] S. Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). http://bitcoin.org/bitcoin.pdf
[52] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 435–464.
[53] Rafael Belchior, Peter Somogyvari, Jonas Pfannschmid, André Vasconcelos, and Miguel Correia. 2022. Hephaestus: Modelling, analysis, and performance evaluation of cross-chain transactions. (Sep. 2022). DOI: http://dx.doi.org/10.36227/techrxiv.20718058.v1

[54] Juan Benet. 2014. IPFS-content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561* (2014).
[55] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. 2018. Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044* 160 (2018).
[56] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. 2021. SoK: Auditability and accountability in distributed payment systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12727 LNCS (June 2021), 311–337. DOI: http://dx.doi.org/10.1007/978-3-030-78375-4_13 ISBN: 9783030783747 Publisher: Springer, Cham.
[57] Pingcheng Ruan, Yaron Kanza, Beng Chin Ooi, and Divesh Srivastava. 2022. LedgerView: Access-control views on Hyperledger Fabric. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD'22)*. Association for Computing Machinery, New York, NY, USA, 2218–2231. DOI: http://dx.doi.org/10.1145/3514221.3526046
[58] Yong Zhang, Songyang Wu, Bo Jin, and Jiaying Du. 2017. A blockchain-based process provenance for cloud forensics. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2470–2473.
[59] Aleksey K. Fedorov, Evgeniy O. Kiktenko, and Alexander I. Lvovsky. 2018. Quantum computers put blockchain security at risk. (2018).
[60] William Buchanan and Alan Woodward. 2017. Will quantum computers be the end of public key encryption? *Journal of Cyber Security Technology* 1, 1 (2017), 1–22.
[61] Tiago M. Fernandez-Carames and Paula Fraga-Lamas. 2020. Towards post-quantum blockchain: A review on blockchain cryptography resistant to quantum computing attacks. *IEEE Access* 8 (2020), 21091–21116.
[62] Roger A. Grimes. 2019. *Cryptography Apocalypse: Preparing for the Day when Quantum Computing Breaks Today's Crypto*. John Wiley & Sons.
[63] Xiaohui Yang and Wenjie Li. 2020. A zero-knowledge-proof-based digital identity management scheme in blockchain. *Computers & Security* 99 (Dec. 2020), 102050. DOI: http://dx.doi.org/10.1016/j.cose.2020.102050
[64] John D. Blischak, Emily R. Davenport, and Greg Wilson. 2016. A quick introduction to version control with Git and GitHub. *PLoS Computational Biology* 12, 1 (2016), e1004668.
[65] Hyperledger. 2019. Hyperledger Besu Ethereum client - Hyperledger Besu. (2019). https://besu.hyperledger.org/en/stable/
[66] Tianyi Qiu, Ruidong Zhang, and Yuan Gao. 2019. Ripple vs. SWIFT: Transforming cross border remittance using blockchain technology. *Procedia Computer Science* 147 (2019), 428–434.
[67] Ryan Henry, Amir Herzberg, and Aniket Kate. 2018. Blockchain access privacy: Challenges and directions. *IEEE Security & Privacy* 16, 4 (2018), 38–45.
[68] Guangsheng Yu, Xu Wang, Kan Yu, Wei Ni, J. Andrew Zhang, and Ren Ping Liu. 2020. Survey: Sharding in blockchains. *IEEE Access* 8 (2020), 14155–14181. DOI: http://dx.doi.org/10.1109/ACCESS.2020.2965147
[69] Ruoyu Yin, Zheng Yan, Xueqin Liang, Haomeng Xie, and Zhiguo Wan. 2023. A survey on privacy preservation techniques for blockchain interoperability. *Journal of Systems Architecture* 140 (2023), 102892. DOI: http://dx.doi.org/10.1016/j.sysarc.2023.102892
[70] Ming Li, Jian Weng, Yi Li, Yongdong Wu, Jiasi Weng, Dingcheng Li, Guowen Xu, and Robert Deng. 2021. IvyCross: A privacy-preserving and concurrency control framework for blockchain interoperability. *Cryptology ePrint Archive* (2021).
[71] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 839–858.
[72] Meng Li, Yifei Chen, Liehaung Zhu, Zijian Zhang, Jianbing Ni, Chhagan Lal, and Mauro Conti. 2022. Astraea: Anonymous and secure auditing based on private smart contracts for donation systems. *IEEE Transactions on Dependable and Secure Computing* (2022).
[73] Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang. 2020. BlockMaze: An efficient privacy-preserving account-model blockchain based on zk-SNARKs. *IEEE Transactions on Dependable and Secure Computing* 19, 3 (2020), 1446–1463.
[74] Rafael Belchior, Dimo Dimov, Zahary Karadjov, Jonas Pfannschmidt, André Vasconcelos, and Miguel Correia. 2023. *Harmonia: Securing Cross-Chain Applications Using Zero-Knowledge Proofs*. DOI: http://dx.doi.org/10.36227/techrxiv.170327806.66007684/v1 Citation Key: BelchiorHarmoniaSecuringCrossChain2023.
[75] Yannis Katsis and Yannis Papakonstantinou. 2009. View-based data integration. In *Encyclopedia of Database Systems*. Springer US, 3332–3339. DOI: http://dx.doi.org/10.1007/978-0-387-39940-9_1072
[76] Alon Y. Levy. 2000. Logic-based techniques in data integration. In *Logic-Based Artificial Intelligence*. Springer US, 575–595. DOI: http://dx.doi.org/10.1007/978-1-4615-1567-8_24
[77] Gilbert Verdian, Paolo Tasca, Colin Paterson, and Gaetano Mondelli. 2018. Quant overledger whitepaper. *Release* V0 1, (2018), 31.
[78] Blockdaemon. 2024. Ubiquity. (2024). https://blockdaemon.com/platform/ubiquity/
[79] Gavin Wood. 2016. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper* 21, 2327 (2016), 4662.
[80] Polkadot. 2021. Cross-Consensus Message Format (XCM) · Polkadot Wiki. (2021). https://wiki.polkadot.network/docs/learn-crosschain
[81] Jae Kwon and Ethan Buchman. 2019. Cosmos whitepaper. *A Netw. Distrib. Ledgers* (2019).
[82] Martin Hargreaves, Thomas Hardjono, and Rafael Belchior. 2023. *Secure Asset Transfer Protocol (SATP)*. Number draft-ietf-satp-core-02. https://datatracker.ietf.org/doc/draft-ietf-satp-core Citation Key: HargreavesSecureAssetTransfer2023a.
[83] Andrea M. Rozario and Chanta Thomas. 2019. Reengineering the audit with blockchain and smart contracts. *Journal of Emerging Technologies in Accounting* 16, 1 (2019), 21–35. Publisher: American Accounting Association.

[84] Yongrae Jo, Jeonghyun Ma, and Chanik Park. 2020. Toward trustworthy blockchain-as-a-service with auditing. In *ICDCS*. DOI: http://dx.doi.org/10.1109/ICDCS47774.2020.00068

[85] Angtai Li, Guohua Tian, Meixia Miao, and Jianpeng Gong. 2021. Blockchain-based cross-user data shared auditing. *https://doi.org/10.1080/09540091.2021.1956879* (July 2021), 1–21. DOI: http://dx.doi.org/10.1080/09540091.2021.1956879 Publisher: Taylor & Francis.

[86] Blockdaemon. 2021. Introducing Blockdaemon's New Staking Dashboard. (Sept. 2021). https://blockdaemon.com/blog/introducing-blockdaemons-new-staking-dashboard/

[87] Iulia Mihaiu, Rafael Belchior, Sabrina Scuri, and Nuno Nunes. 2021. *A Framework to Evaluate Blockchain Interoperability Solutions*. Technical Report. TechRxiv. DOI: http://dx.doi.org/10.36227/TECHRXIV.17093039.V2

[88] Guy Zyskind, Oz Nathan, and Alex Pentland. 2015. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*. https://arxiv.org/abs/1506.03471

Received 14 January 2024; accepted 22 January 2024