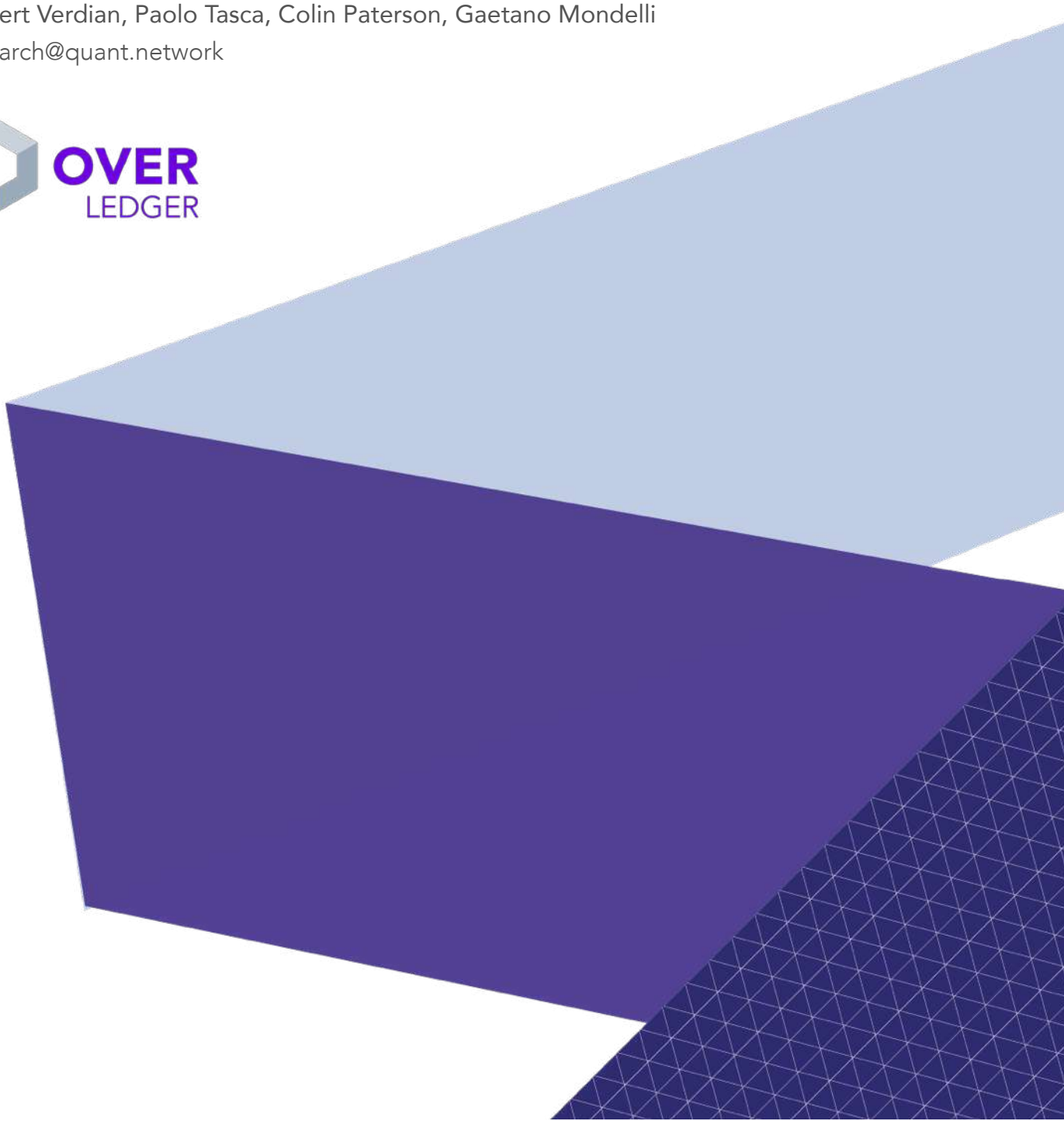
QUANT

Quant Overledger ®
Whitepaper

Release V0.1 (alpha), 31 January 2018

Gilbert Verdian, Paolo Tasca, Colin Paterson, Gaetano Mondelli
research@quant.network

OVER LEDGER



Abstract

The traditional Internet architecture hasn't yet achieved its vision of an open, trusted and secure network. The technology of distributed ledger technologies (DLT) and their broad range of applications across finance, healthcare, supply-chain and many other sectors, is an opportunity for the Internet to adapt and honour what was originally envisaged, and reach its potential as a decentralised network of networks. While providing a broad range of benefits, the fast-paced environment of DLTs lack seamless inter-communicability, internally among ledgers, and externally with existing networks. This limitation forces distributed applications to be single-ledger-dependent, i.e. limited to be only executed on a single ledger. From a technology perspective, the fact that distributed or decentralised applications are single ledger-dependent, makes it difficult to choose the appropriate DLT fit for a purpose, meeting technical and business requirements when considering DLTs for decentralised applications. This technical dependency slows down mass-adoption, limits scope, impedes scalability of new features and reduces necessary controls required for data security and privacy. From a business perspective, a single-ledger DLT application increases the amount of risk, complexity and effort needed to adopt multiple DLT technologies, in many cases even duplicating resources and investments. Enterprises are forced to accept financial risk, due to a monopolistic approach to DLT fees, by limiting options to manage fees and transaction costs. All challenges listed limit widespread adoption and a critical mass of users. This paper proposes a solution to this problem of single-ledger dependency, by introducing a new technology for the design, deployment and execution of multi-ledger decentralised applications. This technology's called Overledger.

Table of Contents

1. Introduction

The invention of Bitcoin [1] achieved a new technological milestone: the blockchain. Although the underlying technology in blockchain isn't new, the innovation results from a combination of existing technologies integrated in an original way, to create blockchains. The components (i.e., Merkel Tree, concatenated hashes, public-key encryption) were established technologies, well before the Bitcoin paper by Nakamoto in 2009 [2].

At the moment of writing, we may argue that, according to the Technology Life Cycle theory, we're at the beginning of the so-called phase of "fermentation", characterised by technological uncertainty, due to the evolution of blockchain into alternative technical paths. The industry promotes different model designs, favouring functional and performance aspects over others, to meet specific business goals. Currently, there are more than a thousand digital currencies (for clarity the phrase digital currency and cryptocurrency will be used interchangeably through the paper) and tokens, and thousands of blockchain projects under development in different sectors worldwide [3].

The current blockchain ecosystem is too fragmented and complex, with progress being achieved in silos. The variation of blockchain designs and their possible configurations represent a hindrance for enterprises, software architectures and developers. A heterogeneous development brings lack of blockchain interoperability and compatibility within blockchain networks as well as existing systems and networks. That is, the ability for blockchains to exchange information between them and with off-chain systems. This lack of interoperability is a threat to the wide and uniform adoption of blockchain applications in our techno- and socio-economic systems. Apart from the technical challenges existing (e.g., key management, filtering and ordering of messages), the lack of interoperability between ledgers, in itself, also presents business risks. A major concern consists of decoupling different business logic from the underlying ledgers to increase the communicability among them by preserving privacy requirements [40-41].

Thus, as was the case with the Internet, it's necessary to provide off the shelf software and development tools that simplify how to develop and deploy distributed ledger (DLT) applications without requiring developers to build various components from scratch. By addressing the above risks, decentralised applications will grow and become increasingly interoperable and widely adopted.

1. Background Information

In order to address the problem of blockchain interoperability, software reference architectures for blockchain interoperability should be set up. However, the majority of existing technologies trying to connect DLTs define a standard of interoperability within their platform, but not outside. The integration with legacy or other DLTs is challenging and difficult to implement. Choosing the correct technology becomes crucial because it's hard to foresee which will be the most suitable, even in the short term, without the uncertainty presented by other external factors such as technologies forking, becoming insecure, or being abandoned altogether. In the development of blockchain applications, the choice of the underlying blockchain technology can't be easily undone. Migrations aren't always possible because the transactions only have scope on their blockchain address space. For example, Blockstack [10], an open-source project to create, manage and use decentralised apps on a blockchain, decided to move from Namecoin to Bitcoin because the first technology was considered less secure. The migration of applications is a problem that can soon apply to those running on Ethereum [17]. This is an endless problem and the solution can't be merely designing better and smarter blockchains. Front-end technologies of today will become obsolete in a few years, due to planned and unplanned obsolescence. To solve these issues, Overledger presents the following solutions:

- Introduction of new scripting languages, or updating existing ones;
- Scaling applications (e.g. increasing transaction speeds);
- Communication among different DLTs for cross-ledger operations;
- Adaptable technology that can change to meet newer sector-specific requirements and regulation; and
- Mitigation tools against new security threats.

Overledger is a new blockchain operating system intending to solve the problems of single-ledger dependency by increasing communicability among DLTs, allowing general purpose applications to run on top of different blockchains. Overledger abstracts single-ledger dependent technology to overcome the bound of different architectures regardless of addresses, ledger implementation and consensus mechanisms. Rather than defining a specific platform with multiple connectors, accommodating the plugin of the compliant DLTs, our solution introduces a vendor-independent wire-level protocol for message-oriented middleware. By decoupling the Transaction Layer with a shared Messaging Layer, Overledger provides a unique solution to interoperability for digital ledger environments. Overledger also allows the business logic to decouple from the underlying ledger. It increases communicability among chains with the privacy constraints decentralised applications demand.
Readers familiar with distributed systems, are aware of the differences between blockchains and the larger family of DLTs. By DLT we generally mean a database technology where records of decentralised and transactional data are stored in sequence (not necessarily grouped in blocks), in a continuous ledger spread through a network across multiple locations. Blockchain is a particular kind of DLT in which batches of transactions are held in blocks, and the blocks are linked with hash pointers in a chain. Each block contains the hash of the prior block in the chain, as a method of keeping the integrity of each set of data in the blockchain. For the sake of

simplicity, here out we'll use the terms blockchain and distributed ledger technologies indistinguishably. From time to time we'll refer to the blocks of the chain. Nevertheless, Overledger is a universal, and general-purpose operating system, among the family of distributed ledger technologies.
Privacy constraints and data ownership in applications requiring transparency is a challenge many projects are exploring (e.g. Blockstack, Digital Holding Asset [14] and Sidechains [5] (Section 1.2)). In the future, when decentralised blockchain applications will spread widely, it'll be necessary to create a method for allowing applications to communicate, exchange and replicate data across multiple blockchains. Finally, it will be easier to compare the performance of different blockchains running the same application. All these aspects together will help boost blockchain technology, its applications, and the opportunities to exploit the technology widely. The next subsection will analyse existing solutions to the problem of single-ledger dependency. We'll also introduce mathematical models with the order theory for blockchains, considering other technical aspects of the underlying blockchain technologies required to deliver such a solution.

2. State of the Art

This section provides an overview of those projects that, although pursuing different final goals, propose alternative solutions to the problem of the lack of blockchain communicability and interoperability. This is a rather new and evolving stream of research; therefore, we focused only on the main projects and compared them with Overledger:

- Virtualchain by Blockstack [7]
- Sidechain by Blockstream [5]
- Interledger Protocol by W3C [15]
- Cosmos by Interchain [8]
- Polkadot by Parity Technologies [16]
- AION-1 by Nuco [19]

We decided to explain the state of the art of communicability among blockchains by using a set of parameters of interest to our project (see Table 1):

1. Purpose;
2. Interoperability;
3. Layer of communication;
4. Connection Method;
5. Connection Speed;
6. Scalability; and
7. Fault tolerance.

1.2.1 Purpose

Virtualchain. Blockstack introduced Virtualchain to build state machines on top of the underlying blockchains. The main idea behind Virtualchains is to extend the blockchain's business logic and allow for the ability to migrate among DLTs, without changing the underlying DLT. Virtualchains are overlays on top

of specific blockchains for building multiple state machines and allowing the migration from one blockchain to another. Migration requires two ledgers to communicate with each other. In this context, some applications run on a single blockchain and are represented by virtual state machines. The way these state machines evolve is by adding new operational codes of functions (op_code) in the transactions' blocks. This is implemented in the additional data field of Bitcoin called OP_RETURN. In this additional field the hash of the new state is added to the state machine after this last operation is executed. It makes the system fork-consistent.

	Interledger	Virtualchain	Sidechain	Cosmos	Polkadot	Aion	Overledger
Purpose	Payments across different payment systems based on DLT	Ability to migrate from one DLT to another for fault tolerance	Add new innovative features to the main crypto currencies	Overcome Blockchain limits and transfer assets	Transfer assets and data (smart contract)	Solve Blockchain isolation problem	Build a messaging layer for multi-ledgers applications
Interoper-ability	1-C-1	1-C-1	1–1	N-C-N	N-C-N	N-C-N	N–N
Layer of comm.	Transaction Level	Over the transaction level	Transaction Level	Protocol based (transaction level for legacy ledger)	Protocol based (transaction level for legacy ledger)	Protocol based (transaction level for legacy ledger)	Over the transaction level
Connection Method	Two phase commit	Two phase commit like	Two phase commit (two way peg-SPV)	Two phase commit like (Tendermint)	Two phase commit (SPV)	Two phase commit	Two phase commit
Connection Speed	Notaries / Entity consensus	Migration time	Confirmation and Contest Period	Proportional to the validator number	Protocol time depended	Protocol time depended	Protocol (flexible) time depended
Scalability	Ledgers allow connectors to run nodes	Ledgers allow to write metadata	Ledger's compliance with two-way peg	Should implement IBF to talk with The Hub	Should implement the polka dot security consensus	Aion-compatible	Ledger's readability and/or writability
Fault tolerance	Depends on notaries or institutions that validates transactions	Depends on the two blockchains involved in the migration	Security faults on sidechain are confined in the sidechain itself.	Confined in the zones (User responsibility on where they move coins)	Para chains follow rules and consensus of Polkadot	Compatible blockchain Aion-1 follow rules and consensus of Aion-1	Protocol based

Table 1: Comparison among projects aiming to increase intercommunication between DLTs. In the Interoperability section below, we explain the convention used in this table.

Cosmos. Cosmos is a project with an ambitious mission: the creation of a network of distributed ledgers that will solve "long-standing problems" in the cryptocurrency and blockchain communities. The Cosmos project wants to solve all the traditional issues of blockchains like energy inefficiency (PoW), limited performances regarding transactions compared to conventional centralised systems, poor governance strategies, for example, when a change in the protocol needs to be implemented.

Polkadot. Polkadot's vision is to build a web where people have direct control of their data. Polkadot, like Cosmos, is a network of ledgers that allows the exchange of assets and data through ledgers.

Interledger. The protocol for Interledger payments, a Ripple project, wants to allow secure payment transfers between different ledgers. Today this service is provided by centralized systems, like Western Union, which need to be trusted by the endpoints of the transactions. Interledger wants to enable payments between any users holding assets in separate ledgers, without the need to trust a third party.

AION-1. Aion seeks to solve scalability, privacy, and communication issues among digital ledgers. It aims to provide decentralised accountability among blockchain networks. AION claims to introduce a next-generation blockchain that enables intercommunicating blockchains. The first generation presents the distributed ledger technology, cryptographically secured like Bitcoin, and the second one enhances smart contract and distributed apps, like Ethereum. According to the AION classification, the third generation adds the ability of communication and value exchange among ledgers.

Sidechain. Sidechain and Interledger protocols are two projects aiming to promote communication among blockchains and allow asset transfers. While Sidechain builds ad-hoc external chains that can interact automatically with just a specific blockchain (not full mesh connection), Interledger needs the use of external nodes (the connectors), with a further consensus mechanism to allow currency conversion, without the need to trust a third-party exchanger. The focus of Sidechain's to allow ledgers (notably Bitcoin) to transfer coins in particular chains that are innovative and have new cryptocurrency features. This should extend traditional cryptocurrencies like Bitcoin, limited by their original protocol.

Overledger. The projects introduced so far are "single-ledger dependent". They start from the same point: blockchain protocols are not adaptive to needs. I.e., they propose particular solutions to solve only specific needs. Differently, Overledger is a universal and general-purpose technology. For this reason, although Overledger shares common technical aspects with some of the existing projects, it decouples the business logic from the underlying ledger technologies. This means general-purpose applications can sit on top of different ledgers at the same time, with the ability to communicate. In fact, we propose an "over layer" on top of existing blockchains which applications can run on. Thus, the applications can communicate, migrate and exchange information and value, regardless of the ledgers on which they've been deployed. Overledger enables users to build decentralized multi-chain applications which aren't single-blockchain dependent. This means users can run applications, smart contracts, treaties or move data across different blockchain technologies. This original approach will empower the adoption of blockchain technologies across various sectors and use cases, enabling large-scale adoption of the technology, without tying it to a particular vendor or chain.

1.2.2 Interoperability

One of today's challenges with distributed ledgers is they exist in silos. Different structures and working mechanisms make it harder, if not impossible, to build a common interface. Moreover, in general, users have different rights on different ledgers. For example, only a group of people can append data on a permissioned ledger while another one can only read the data. We tackle the interoperability issues by positioning Overledger on top of the ledgers, rather than struggling to order and match different ledgers. In our approach, we distinguish ledgers with the ability to read and write from ledgers that are readable only. A ledger that offers a write function of arbitrary strings long enough to host a hash is classified as writable. In this case, users can use optional fields of a regular transaction to add the hash of arbitrary messages. The ledgers that don't offer a write function of arbitrary strings, but do offer a read operation, are labelled readable.
Until now, the proposed solutions for blockchain interoperability depend on the specific goals for which the projects were originated.

Interledger. The protocol for Interledger payments can connect any two ledgers having a connector with at least one node in each of the ledgers. It's possible to connect two ledgers, let's say A and C, not directly joined by a connector, if there's a path between ledgers that starts from A and ends in B.

Virtualchain. Virtualchain allows running a state machine on the top of a specific blockchain. If the underlying blockchain becomes obsolete, or no longer suitable, Virtualchain offers the ability to migrate to a more convenient one. For this reason, the interaction between blockchains happens only between the two blockchains involved in the migration.

Sidechain. In Sidechain the communication happens between two chains, the main one called the Parent chain and the other one with new features called the Sidechain.
Newer applications can use existing blockchains for which security is consolidated as a lower or parallel layer of their business logic. These applications can build messages out of the chain, hash them and add only their digests in the blockchains. In the case of parallel chains, they can build these extension-chains that allow the exchange of assets with the main ones. The extension chains provide benefits, allowing the development of more complex applications and delegating the complexity in the external layer. The Sidechains project uses this approach to transfer assets between existing blockchains like Bitcoin, Litecoin and others offering new features. These extensions should always be compliant with the main blockchain protocol (e.g. the Bitcoin one) without changing it, because the protocol isn't flexible. Even small changes would result in a hard fork.

Cosmos. Cosmos has an architecture that allows zones (blockchains compliant to the protocol) to communicate to each other through hubs, multi-asset blockchains, where coins can be owned by individuals or by zones. The Hub blockchain appends transactions to its ledger among individuals or zones.

Polkadot. Polkadot uses relay chains to coordinate consensus among parachains. Parachains gather, collect and process transactions and use the relay chain consensus for normal and cross-ledger transactions.

AION-1. Aion provides a protocol to route, propagate and translate messages among blockchain networks. At the root of this network there's an ad-hoc blockchain called Aion-1.

Overledger. It should be noted that all of the above solutions only allow communication between two blockchains at the same time. Our solution can conduct operations across multiple blockchains simultaneously. Overledger can read information like transactions, scripts or contracts and map them in the "over layer", only if that information is compliant with the selected business logic. In this way, Overledger can connect all blockchains allowing the ability to add an arbitrary hash of a message to a host. In most cases, the projects that encourage connectivity among blockchains create a network of ledgers, where the asset, or the information, needs to be routed from the source to the destination, across different hops (when it is applicable). We do not propose a network approach; we introduce a multi-layer approach. We move the information in the layers above, creating a common interface among ledgers.

In table 1 we summarise the level of interoperability by using the following notation:

- 1-C-1: Two connected blockchains per time with a connector;
- N-C-N: Many connected blockchains per time with connectors;
- 1–1: Two connected blockchain connected per time without connector; and
- N–N: Many connected blockchains per time without connectors.

1.2.3 Layer of Communication

Our idea's to build a presentation layer on top of blockchains to allow applications to run on them. Communication among blockchains happens on a logical layer, which sits on top of the transactional one. Virtualchains adopt a similar approach to ours. Virtualchains are designed to perform migration between two blockchains. The migrations are executed through messages exchanged on a virtual layer. This means the transaction where the fingerprint of the operation is added doesn't affect the underlying transaction. Instead, the other projects carry out the communication at the transaction level. Since the protocol for Interledger and Sidechains focus on tokens and coins, they need to impose the connection happens at the same level of transactions. Cosmos, Polkadot and Aion define new standards to work on a new sharable transaction layer. A limitation of this approach is that they can only work with token-based ledgers.

1.2.4 Connection Method

All the projects in Table 1 propose different ways to communicate across-ledgers.
However, when it comes to interactions with different ledgers, in all cases, connection methods between different ledgers require a two-phase commit approach. This consists of two or more states. The first, where the transaction is proposed, propagated and processed (committed or aborted) by the stakeholders.

Interledger. In the case of the protocol for Interledger, particular nodes called connectors are in charge of communications. A connector of two chains is an entity having at least one node in each of the two ledgers. They coordinate transfers on multiple ledgers using smart contracts. Smart contracts force connectors to cooperate, therefore people don't need to trust anyone. The escrow's implemented with the financial equivalent of the two-phase commit, that first locks resources and then moves them, or rolls back the transaction to the previous owners.

Virtualchain. Virtualchain's migration uses a two-step commit that locks the application in the new blockchain until the migration is ended.

Sidechains. Sidechains are newer blockchains that have particular rules allowing transfers from parent chains. They proposed a scheme that locks coins on one chain by sending them to unique addresses, then a simple payment verification (SPV) proof is used to unlock them on the other blockchain. SPV are used by lightweight clients, but rather than checking the validity of all transactions in the blocks, they only check a transaction's a member of the Merkle tree in a block appended in the past. This method's described in the original Bitcoin whitepaper [1]. This is much faster, as users only need to check the membership in a Merkle tree and only the block header. Only when the block header is received the transaction that belongs to the longest blockchain is checked, as can be seen. This doesn't guarantee it's the longest valid chain.

Cosmos. In Cosmos, the zones communicate with each other through the "Hub" with an Inter-Blockchain Communication (IBC) protocol defining two types of messages (IBCBlockCommitTx, IBCPacketTx). This is similar to the SPV approach.

Polkadot. Polkadot seeks to address the unsolved questions of extensibility and scalability by decoupling the tied bound of canonicality and validity in the popular blockchain architectures, with a focus on security. Parachains communicate among them through the relay chain. The relay chain provides the foundation (through a consensus mechanism, a parachain interface and a routing protocol) on which the Polkadot network's built.

AION-1. Aion, like Cosmos and Polkadot, connect blockchain networks compliant with a set of requirements (AION-compatible) and make them communicate through a central node, in this case, the AION-1.

Overledger. Also, Overledger adopts a two-phase commit schema, similar to the one used by the other projects.

1.2.5 Scalability

Interledger. The scalability of the payments for the Interledger protocol depends on the connectors. If a connector can run a node in a ledger, having the right to do transactions, then it can potentially use this protocol to exchange to and from the particular ledger.

Virtualchain. The scalability for virtualchain depends on the ability of a blockchain to add metadata on the blockchain.

Sidechain. For Sidechain, the scalability parameter depends on the blockchain's ability to implement a two-way peg scheme. It must be considered that the two-way peg scheme uses SPV proof, that requires proof of work, which means it needs to be adjusted for blockchains using another consensus mechanism.

Cosmos. Cosmos can scale as long as a new blockchain can adopt the inter-blockchain communication protocol and also connect to a Cosmos Hub.

Polkadot. Similar to Cosmos, Polkadot can scale if new ledgers adopt the requirements to connect to a relay chain, i.e. new ledgers must be compatible to the connector standard.

AION-1. AION-1 also follows a similar approach to both Cosmos and Polkadot.

Overledger. Internet-scalability has been built into the core of the Overledger protocol and approach.

1.2.6 Fault Tolerance

Interledger. For the Interledger protocol there's two modes of working: Atomic mode and Universal mode. The fault tolerance in the atomic mode depends on the behaviour of the group of notaries that approve transactions. In the case of universal mode, it depends on the institution that fills that role in charge of an incentive. Since the Interledger protocol is only intended for value exchanges among blockchains, a blockchain error can't compromise the business logic. If a blockchain becomes insecure, connectors won't transfer values from or to that blockchain.

Virtualchain. Fault tolerance is the main reason behind Virtualchain, if a blockchain becomes insecure, the application can migrate from one blockchain to another. However, if a blockchain's already fully compromised, the previous state of the application can't be recovered. This is very unlikely to happen, but to avoid this, Virtualchain frequently publishes logs on other blockchains.

Sidechain. With regards to fault tolerance, one should consider that Sidechain treats the two chains as isolated. In the case of a security fault or a malicious design in a sidechain, the damage is confined to the sidechain itself and doesn't compromise the main one.

Cosmos. In Cosmos' topology, the Hub is the critical point of failure, therefore security is enforced. The Hub uses the Tendermint consensus protocol [18], considered secure, and uses only trusted validators for the transactions. The problem arises if one of the zones has a security issue, it can spread through the network. The Hub doesn't validate transactions inside zones, it only validates inter-zone transactions.

Polkadot. Polkadot also has a single point of failure in the relay chain. This is the reason the project focuses more on security for these entities.

AION-1. In the Aion-1 blockchain, at the root of the network, is a potential single point of failure for this type of technology.

1.2.7 Connection Speed

Although it's very hard to estimate the connection time, let's see how the different software architectures of the projects in Table 1 may influence it.

Interledger. The Protocol for Interledger payments operates over three phases: proposal, preparation and execution. The phasing works utilising a two-phase commit protocol, and a chosen group of notaries act like the Transaction manager. The transaction time depends on the time required by the blockchain to add information about these three phases for all nodes involved in the transactions, plus the time required by the notaries to check all the information needed at each phase of these protocols, to send a vote

(commit, or abort). The liveness property (deadlock free and not starvation) is guaranteed in this process because of the presence of timeouts in each stage.

Virtualchain. Virtualchain is a logical (not physical) layer on the blockchain showing the evolution of the state of a virtual machine. This chain needs to respect the consensus rule to be fork-consistent and, depending on these rules, we can give a rough estimation of the speed of their transactions. Where there is communication among blockchains, i.e. migration, they require only two messages: one to announce they're leaving a message on the old blockchain and one on the new blockchain, announcing a migration started on that blockchain.

Sidechain. Sidechain's two-way peg requires sending SPV proof to lock coins on a blockchain (parent or side one) and then unlock them on the other. This process delays a specific period of time before sending the SPV proof (confirmation time) to allow sufficient proof of work to be created, after the SPV proof's been sent (contest period), to prevent double spending. According to the Sidechain white paper [5], this period would be one or two days.

Cosmos. The connection speed in Cosmos depends on the parameter of the Tendermint consensus protocol [18]. A good trade-off with one hundred validators can allow thousands of transactions per second.

Polkadot. In Polkadot the connection speed depends on the protocol and the involved entities (nominators, collators, validators, and fishermen).

AION-1. To estimate the connection speed in Aion we need to consider its architecture. Considering Bitcoin can manage approximately seven to ten transactions per second, given the underlying speed, this will depend on the transactional speed of the underlying Aion compatible blockchains involved, as well as the Aion-1 blockchain itself.

Overledger. All projects that don't create a presentation layer and aim to achieve communication at the transaction level needed to route information across-ledgers from the source ledger to the destination. Since our solution doesn't route the information across-ledgers, the connection time is proportional to the latency of the involved chains and only requires a fixed set of transactions. It doesn't depend on the path length, since all ledgers are directly connected to our presentation layer.

3. Overledger Philosophy

One of the challenges with digital ledgers is they exist in silos. Different structures and working mechanisms make it harder to build a common interface. Overledger addresses this issue sitting on top of them rather than struggling to match them up. Sometimes you may have different rights on different ledgers. Overledger's vision is to build a unique group of transactions, put together, and then ordered by distributed applications. The analysis of the projects in Table 1 on DLT inter-communication and interoperability allows us to extract the connection methods adopted in those cases. A comparative analysis permits us to extract the common logical elements pertaining to those methods:

- Build a platform with new features designed to address today's DLTs challenges (slow transaction rates, migrations, cryptocurrency exchanges etc);
- Build a network of blockchains where nodes have different roles and responsibilities (e.g. parentchain-sidechain, hub-zone, relaychain-parachain [16]);
- Define a standard to connect blockchains to the platform;
- Build adapters to make existing technologies compliant with the standard;
- Transaction-oriented, not application oriented;
- Connections at low-levels strictly involving consensus mechanism; and
- Tree/Graph network that requires complex routing algorithms.

The methods proposed so far are very limited in their application, as they set strict rules and create a standard on the connection level. This approach automatically excludes some technologies and limits the evolution of the new ones are bounded within the new constraints. Moreover, it becomes harder and harder to include the current and new technologies being developed. It's indeed impossible to build a general meta-adapter to connect all present and future blockchains, resulting in a platform needing to build different adapters of different complexity. Fig.1 represents a common scenario where there's special nodes with more responsibility than regular ones. The special node, becomes a single point of failure for the connected bodies. It also needs to be able to manage as many nodes as possible. If it's designed to be optimised for the connection: this is represented with a hexagon. The hexagon is the symbol for connectivity because of its property to cover a plane (its internal angle is a divisor of 360). Despite its design, it needs to connect legacy or existing technologies not designed for inter-communicability (the purple and the blue polygons). Therefore, it must implement and maintain different adapters. The yellow node is the representation of a brand new blockchain technology that's particularly complex, and therefore its adapter has an irregular shape to highlight its complexity.

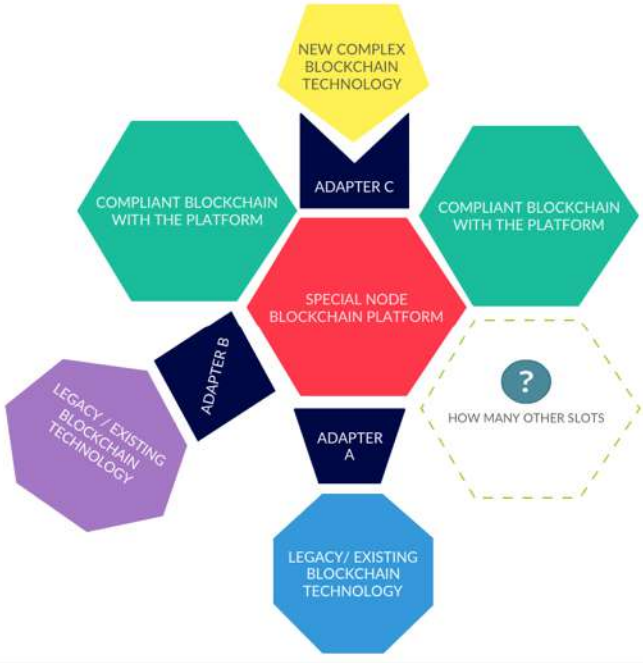


Fig. 1: Connectivity and differences between regular and special nodes.

Other issue are upgrades or changes. The upgrade of the communication protocol, or standard, adds complexity. Since we're dealing with transactions, it can lead to unavailability of the service or inconsistency problems. This also affects the flexibility of the communication because of the resistance to

changes. Furthermore, one should take into account that an upgrade of the platforms may also require the adapters be updated. The resulting architecture is a graph network requiring different adapters for different DLTs and a routing strategy that forwards information from a source to a destination, avoiding loops. Fig. 2 shows a topology for such an architecture.

Fig. 2: Connection Topology.

Let's suppose DLT-A wants to send a message to DLT-G, the following issues arise:

- Convert the message in the platform format from the DLT-A format respecting the time;
- Route the message from the node to the next one. The routing strategy must be efficient because the different consensus timing adds complexity. The fastest path may not be the shortest one. The safest path may not be the fastest one. The user may not have control of the path of their information;
- The message needs to be converted from the DLT-A platform format to the DLT-G one; and
- Special node (master, relay, parent, hub) are points of failure as well as the adapters itself, in this example.

The Overledger approach manages the connection in an upper layer despite the underpinning technology. The applications bind blockchains and treat them as a decentralised queue of transactions where messages are attached to transactions and are appended to the different ledgers.

2. Overledger

In this paper we propose an architecture that acts like the OSI model for the Internet, redistributing the tasks among four different layers, built on the transport layer, since most blockchain technologies are built on the Internet (see Fig. 3):

Transaction Layer. This layer stores transactions appended on the ledger's technology. It includes all operations needed to reach the consensus in different blockchain domains (in this representation, we simplified this by putting all those operations in one layer). However, all the transactions executed on a specific blockchain only have scope in that domain, i.e. it's not possible to also make them valid in other ledgers. Therefore, this layer's represented by different and isolated ledgers.

Messaging Layer. This is a logical layer because all relevant information is retrieved from the ledgers. Information can be transaction data, smart contract or metadata (if the underlying ledgers can add arbitrary strings on transactions). In the particular case of metadata, the added strings are typically the digest of out-of-chain messages that can be interpreted as the payload in this logical over layer. This logical layer stores all transaction information and the message's digests of different applications in the same way as a shared channel has packets of different applications.

Filtering and Ordering Layer. In this layer messages extracted and built from the transaction information, and those only referenced in the transaction through a hash, that is exchanged out of the chain, are filtered and ordered. This layer is responsible for creating connections among different messages built in the Messaging Layer. In the case of metadata, this is the layer in charge of the validation of out of chain messages. The validation checks the application schema and its requirements. Application requirements can be set on the transaction data. For example, the application may accept only transactions from/to a particular address, or may need a certain amount of coin to be moved. Therefore, applications can only consider valid messages that move a certain amount of coins to a specific address.

Application Layer. Valid messages that respect the requested format and have the requested signatures from the list of the application's messages. Messages can update the state of their application. Different applications can share the same messages or can refer to messages of another application. The message references are the unique hash pointer to the transaction in the ledger containing the digest of the messages. The hash pointer is basically a pointer to the place where some (cryptographic) hash of the information is stored. It's an identifier that can be used to uniquely select a transaction in a database and to verify it hasn't changed.

2.1 Messaging Layer

There are several reasons we think the control logic, as well as the business logic, should be decoupled from the transaction level. The first reason lies in the consideration that, in general, blockchain protocols can't accept changes without a fork of their chain into two different ones. For several reasons, the blockchain communities can decide to split and depart from the original chain by adopting different protocol rules. For example, the Ethereum platform has been forked into two versions: "Ethereum Classic" (ETC) and "Ethereum" (ETH). Prior to the fork, the token was called Ethereum. After the fork, the

new tokens kept the name ETH, and the old tokens were renamed ETC. Ethereum Classic appeared as a result of disagreement with the Ethereum Foundation regarding The DAO Hard Fork [38]. Other notable examples regard forks of Bitcoin. Reference the implementation of Bitcoin Core, aiming to increase the transaction processing capacity (e.g., Bitcoin XT, Bitcoin Unlimited, Bitcoin Cash) [39].

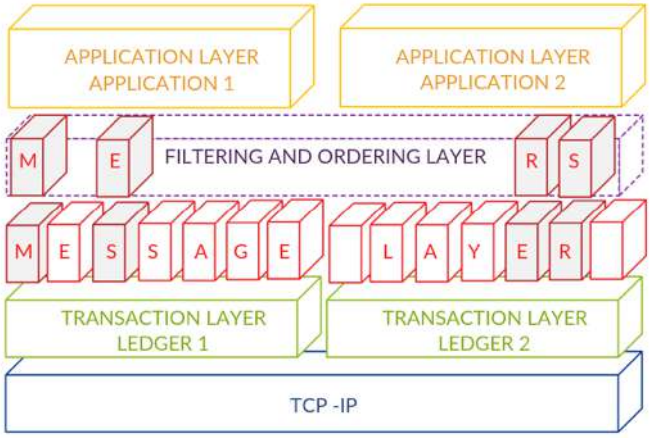


Fig. 3: Overledger Architecture Model

Another fact supporting our architecture choice lies in the fact that Bitcoin, and other blockchain based projects, have originally been conceived and designed to support cryptocurrencies and simple transfers. That's why smart contracts and distributed scripts aren't as powerful as today's application's demands. Moreover, transactions only have scope on their blockchain address space because of the simple original design. Finally, our architecture is justified by the need to be code language independent. Indeed, the languages used to build today's blockchain systems will likely be substituted by other languages in the future. For example, the Bitcoin Script, the script language of Bitcoin takes inspirations from FORTH - a 50-year-old concatenative programming language. Instead, many other blockchains, like Ethereum, have implemented a complex business logic within the blockchain layer. Serpent and Solidity (the two Turing complete programming languages that run on Ethereum) are inspired by Python and JavaScript respectively, the most adopted scripting languages today, but not necessarily of tomorrow. In the remaining part of this subsection we explain the technical requirements the Overledger Messaging Layer needs to meet.

2.1.1 Standardisation

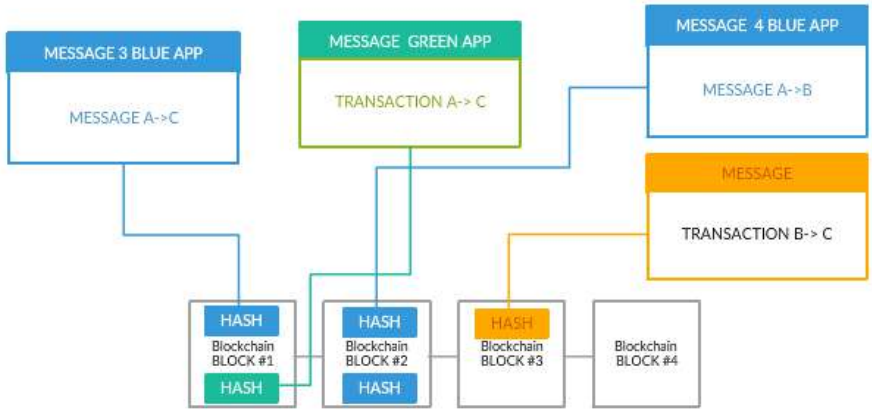
The Messaging Layer shall abstract from all the transactions regardless of the particular ledger. Only the applications (living in the Application Layer) are responsible for the design rules (according to their business logic) that will specify which specific information will need to be extracted from the transactions, and in which order they'll need to be sorted out. The Messaging Layer extracts all the information about the transactions from every single ledger in the Transaction Layer. The Messaging Layer stores all the information recorded in the ledger at any transaction: the source, the recipient, the amount of coin, the script used, the smart contract and all other types of information that can be recorded in each ledger, laying in the Transaction Layer. It allows legacy applications to work with our architecture, extending the capability across-ledgers. However, ledger technologies are very different from each other and follow different architectural standards [29]. This requires us to build a unique standard naming system for the Messaging Layer. The standard should define how to refer to the different ledgers and to their particular fields in a unique and unequivocal way. For example, it should be possible to refer to the address of a

Bitcoin transaction using a compound syntax (e.g. BTC.Height.TransactionIndex.Outputindex should apply to the output at index "Outputindex", of the transaction at index "TransactionIndex", in the block of the main chain, with the height equal to the "Height" parameter.) Our proposal will give the opportunity to build standard libraries for developers to interact in similar ways with different ledgers.

2.1.2 Messages and Information Out of the Chain

Most blockchains provide the ability to add a small amount of metadata to the transaction input/output. For example, Bitcoin allows us to use the OP_RETURN [23] field to add metadata. Ethereum has the field used for the script bytecode [24]. Ripple has the 'memos' field [25]. Therefore, external messages (converted into hexadecimal values) can be used for any arbitrary logic. These messages can be considered transactions, or blocks of an external blockchain, or locally stored by users. Only fingerprints of these messages are added on the blockchain. The Digital Asset Platform [14] uses this approach as a privacy enhancement tool, to solve the reconciliation problem between parties. Each user has its own set of views and can build its view of the transactions. Only the hashes of these transactions are stored in the [public] ledger to prove their validity. In case of conflict between two views, a user can show the validity of their set of transactions by showing the transactions containing the hash of their messages, and prove their validity to the counterpart. We want to extend this approach to messages. Messages are an exchange of information; they can be a simple transaction, or something more complex concerning the application logic.

Fig. 4 depicts an example of our hashing approach, applied to the Messaging Layer. In this example, there are different applications running on the top of the blockchains and their messages are differentiated by using different colours. From the top, the first white blocks represent a section of the "white" blockchain, to which the coloured applications (blue, green, orange) are appending messages. The "white" blockchain only hosts the hash of those messages. Each hash corresponds to a message of one of the applications. There could be more than one message of the same application, in different transactions, stored in the same block and there can be messages of different applications in the same block (see block no. 1 of the white blockchain). Moreover, the same application can run on different blockchains at the same time. This is shown in our example by letting the blue application add some messages, both on the "white" blockchain and some others on the "blue" one (bottom of the figure). Note that the numbers of the messages are monotonically increasing.



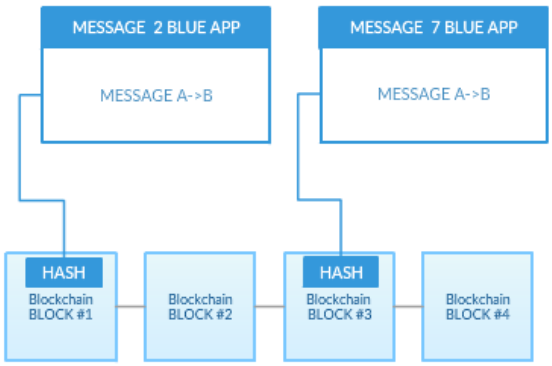


Fig. 4: Messages out of the chain of different applications on different blockchains.

It should be pointed out that the concept of a blockchain "message" is more generic than one of a blockchain "transaction". In fact, messages can require a more complex format and be part of more complicated interactions between the parts. Hence, mark-up languages like XML or JSON can be a good fit to build messages in this context. The applications could define the valid sequences of messages and their format through a schema to validate them. Moreover, these messages can be exchanged over standard HTTP machines and XML, and JSON, all good choices since they're already widely used in Internet communication.

2.1.3 Messages and Security Properties

The security properties of the Messaging Layer regard the solutions to the problems of identification, authentication, authorisation and non-repudiation of the entities and actions involved in the messaging.

Identification. Identification is the action to claim to be a certain entity, e.g. a person, a company, a department with no ambiguity. In a login system, a user identifies herself through the username, usually an arbitrary alphanumeric string. In blockchains, the identification is usually achieved through public keys, or their hash. In this context, a public key seems to be the best approach, but there can also be cases where users are identified with an address which similarly works as a username. In this respect, the message format should always have a field which unequivocally identifies the senders and the receivers.

Authentication and Authorisation. Authentication is the action that proves someone is who they claim to be. In standard login systems, a user can be authenticated with a password. In the blockchain, authentication is done with a private key in the process of the digital signature. In such a context, asymmetric cryptography and digital signature are the common approaches to enable a message compliance with both the will and the authentication of a user. However, after the initial authentication phase, with username-password, it's also possible the application shares a secret with the user, then it starts to exchange messages encrypted with symmetric cryptography. The hash of these messages is stored in the blockchain, and the future user, with the shared secret and the complete message, will be able to prove they've received those messages.

Non-Repudiation. Non-repudiation means that, in the future, an action can't be questioned or disowned to have taken place. Digital signatures are the most common way to achieve this property in computer science. This means that, if someone has a copy of the signed message, it can be stated that the message was compliant with the will of the entities who signed it.

Privacy Constraints. Many applications will need to obfuscate data related to other users in order to be compliant with privacy requirements. The classic lower level of blockchain cannot satisfy privacy constraints because of the intrinsic transparency property of the blockchain. In other words, a common blockchain transaction can't be completely anonymous, otherwise it would be impossible to check its validity. With regards to this, our proposed Messaging Layer borrows a similar approach to the Digital Asset Platform by adding the hash of the messages in a special field of the regular blockchain transaction. If someone holds a copy of a valid message (e.g. signed by all the stakeholders), it has the opportunity to show the hash is in the chain, hence it's valid. Colored Coins (Bitcoin applications for digital representation and management of real world assets) uses a similar mechanism by using the OP_RETURN field of Bitcoin [36]. Thus, the proposed solution may have some drawbacks because everyone can add the hash of a message into blockchain transactions, creating the so called "blockchain bloat" problem [3]. It means there can be irrelevant or invalid messages whose hash is the chain. Therefore, the presence of the hash in the chain isn't a sufficient condition to treat the message as valid, according to the Overledger logic. We cover how to check message validity in Section 2.1.2.

Fig. 5 depicts a situation in which messages are exchanged off-chain, for example, through the Internet. Even if it would be sufficient to save the hashes of the messages in separate transactions, here we represent them stored in different blocks of the chain. User A can see all the hashes in the chain, but she only has details of her own transactions: the blue one and the green one. She, for example, doesn't know there's a transaction between User B and User C. If there's conflict between the users' views of the records, each user can exhibit their complete transaction history and prove the validity of their transactions by sending the hash pointers to the counterpart, such that they can be used to verify the data in the blockchain.

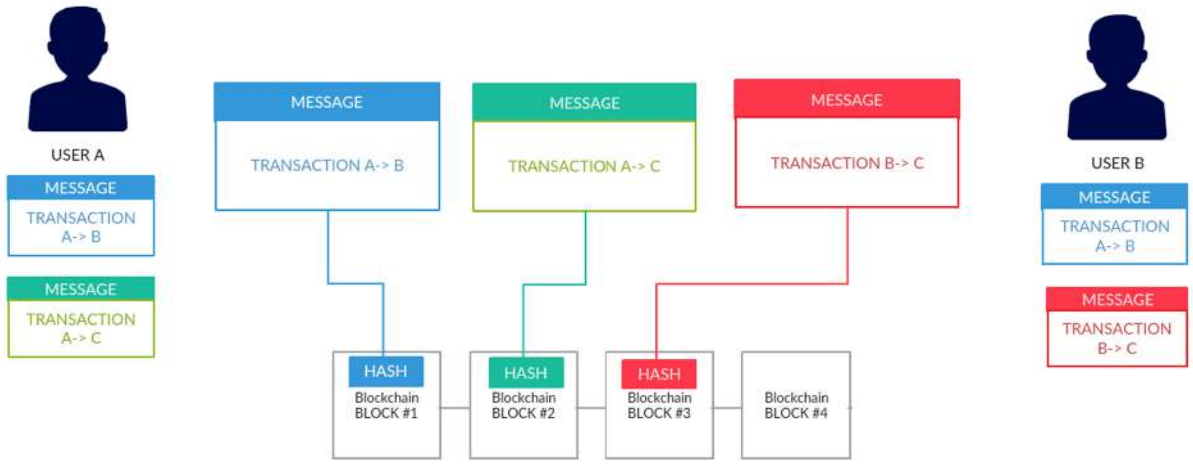


Fig.5: Messages out of the chain and user's views.

2.1.4 Improve Privacy Specification

To enhance user privacy, we conceive the introduction of the multi-part message hash pointer technique. In Subsection 2.1.3 we show a way to give the proper view of the application to the proper entity. However, in certain circumstances the applications may require messages to follow hierarchical structures that may have different scope regarding privacy. Since in our context we have arbitrary message formats, we propose adding hash pointers to another part of the message. In this way, the applications can provide each user with user-specific parts of the messages, i.e., unique content addressed only to specific users. Thus, if a user has no rights to see some part of the multi-part message, they'll only see a hash pointer or a random piece of data. In Fig. 6 we provide an example of messages out of the chain, with different levels of secrecy. In our example, the user with only the right to see the "blue" part of the message, is only aware there are two other branches of the message: the one that starts with the "red" part and the one that starts with the "green" one. However, the user doesn't know how deep these branches are. At the same time, the user that can see the "black" part of the message knows about the "green" one and the "blue" one, but we can't know if she's aware of the content of the "red" part. Please consider this modular approach can be replaced by removing the pointers from the lower part, adding a reference to the upper ones and treating them as newer messages, adding their hashes on the chain. However, this can lead to business logic inconsistency if the other parts of the messages aren't correctly added, or something happened in between. The problem with the approach described above is, whoever sits on a given level of privacy is aware of the other levels of privacy. This is something that should be avoided in most situations. To solve this issue, we can hide the level of secrecy by taking the fields that contain private information like hash pointers, encrypt them with the public keys of the interested user and make it look like a random piece of data to other users. This means, in the case there are no more levels of secrecy, we should add a random sequence of data.

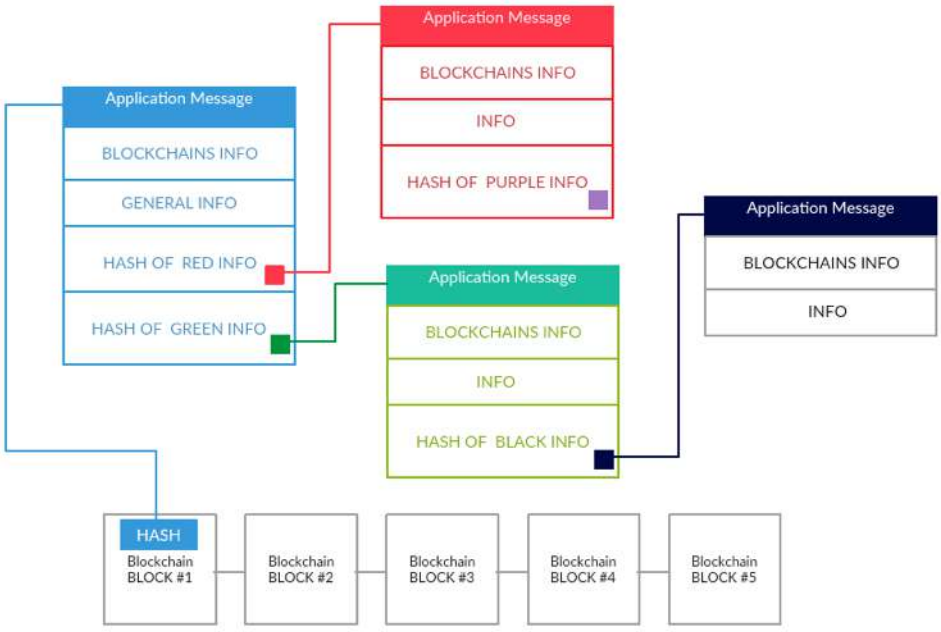


Fig. 6: Messages out of the chain and different level of secrecy.

One of the problems with this approach is the secret information can be different in size and it means we need to amend the size and use a padding strategy. The main issue to be solved is the users' awareness regarding the different parts of the message. Since a message can be encrypted only with one key, if we want to use this strategy, we need to replicate the different parts concerning that specific key. For instance, in Fig. 8 three users can see the "green" part of the message and just two of them can see the "black" one. We need to add three "green" hash pointers to the "blue" extension. One of these three (the one on the top of the figure) has random pieces of data as an extension field. The following ones (the one in the middle and the one in the bottom of the figure) have a hash pointer to "black" messages encrypted with their own keys. Extending this example, we can see the lower the level of secrecy the higher the number of keys involved.

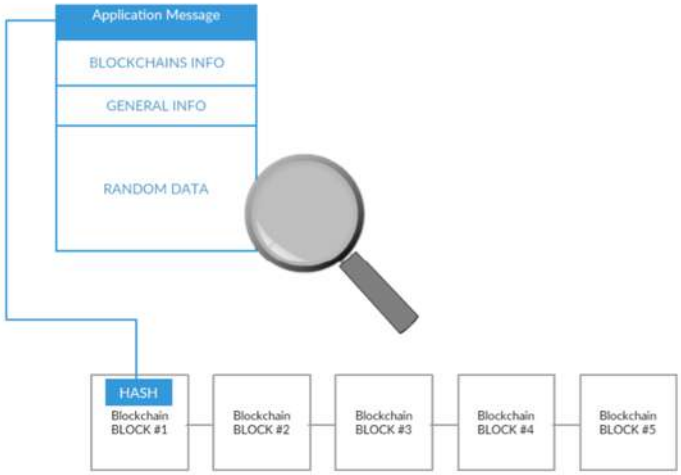


Fig.7: The field to extend the message contains the hash pointers of the other parts of the message and also contains a random sequence of bytes.

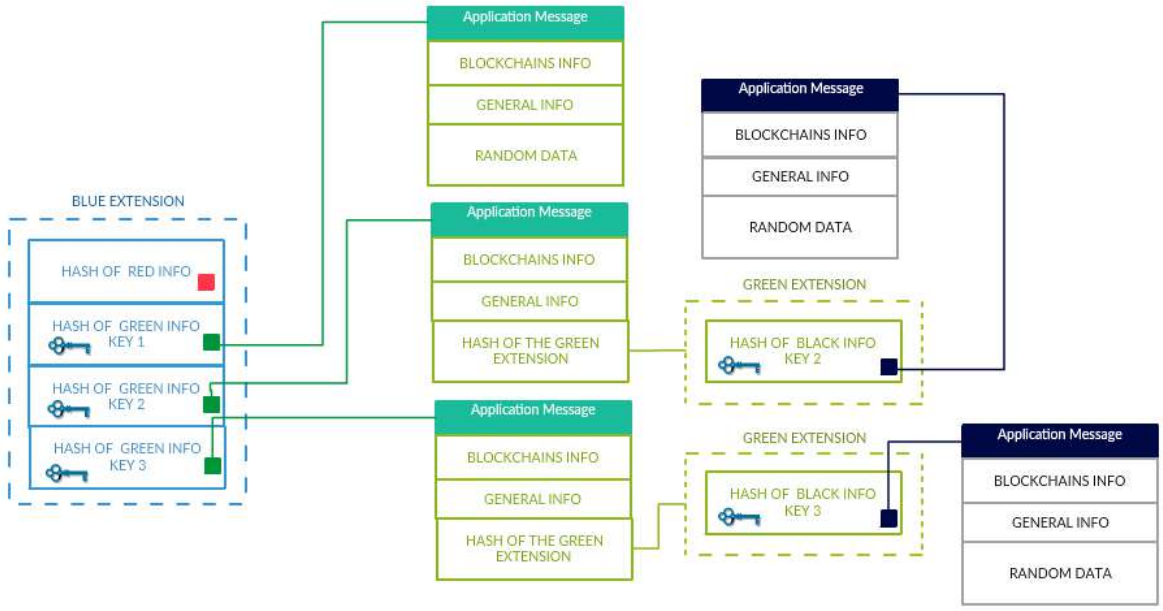


Fig. 8: Cryptography with multi-part messages.

Even if we showed with our example in Fig. 8 that it's still possible to manage multi-part messages with different keys, we still have the issue related to the different message sizes, with the need to fix the size to make sure authorised users aren't aware of how many branches of secrecy there are at that level. In this section, we propose an extension field approach (see Fig. 9), that partially solves that problem by adding only one field per message that can point to a message part that only contains hash pointers. In this case, the "blue" message has only one extra field for the extension. The content of that field can be a random sequence of bytes or the hash of the "blue" dashed extension. The "blue" extension contains the pointers to the "red" part and the "green" part of the message. In this way, users who sit at a lower level don't know if there are other levels of secrecy or how many there are. However, with this solution, the users that can read the "red" part of the message also know the "green" one, as well as those who can read the "green" part also know the "red" one.

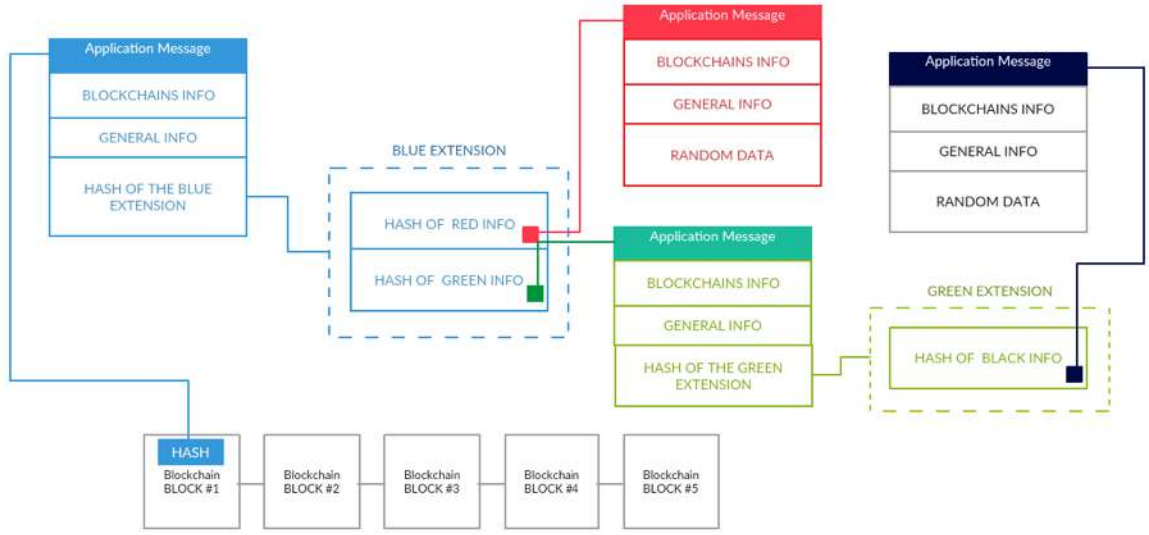


Fig. 9: Level of secrecy without keys.

2.1.5 Shortcomings of Messages Out of the Chain

One of the main issues needing to be overcome when building an over-layer of the message, whose hash is added on the blockchain, is we can't control who and which messages can be appended. It means if there's no rules to limit it, there can be problems like trashing and spamming. It's also possible to design a denial of service. This is what would happen in the lower level, if the blockchain has no consensus mechanism. In fact, the consensus rules how frequently blocks are added in the chain.

Spamming. In a messaging system, spamming is the action to send unsolicited messages. This action results in overwhelming the channel with valid messages. In this case, the channel is the space per transaction of the field we use to add the hash of the application message. One possible solution can be to let only a third party validate messages by signing them. In this case, only messages with the right format, which have been signed by the authority and have hashes in a valid chain, can be considered valid. However, in some business contexts, this approach could be mandatory, and would still give the user the transparency, and non-repudiation property, traditional centralised applications can't guarantee without trust. Without a centralised authority checking message consistency and controlling its growth, replica issues or replay attacks can also occur. For example, a command of a valid message added many times, could be executed an arbitrary number of times. Also, it becomes more difficult to build the application logic if during the scanning of all messages there are messages that have the correct format and the right signatures, but they're no longer valid because they're already in Overledger. There are techniques to solve this issue, like enumerating messages or using timestamps.

Trashing. Trashing is the action of overwhelming a channel with meaningless data regarding the business logic (not-valid messages). It can be the action of a malicious user or an involuntary act. In fact, many applications can run on the same blockchain, and messages from one application are trash data for other applications (i.e., lack of M2M communication). This means if an entity of the application loses the pointer

to the particular block transaction that hosts the hash of the message, it should compare this hash with all the hashes of the valid messages of all the applications that have run into that blockchain, until the matching one's found. It also means, to check if a message has been validated and added to the chain, users have to check all the messages and then store the pointer to it.

2.1.6 Speed and Throughput of Messages

Building new decentralised applications on the blockchain layer are affected by the parameters concerning the underlying layers (e.g. speed of transactions, energy efficiency, scalability). In fact, if we want to act as the "over-layer", we must accept and respect the latency of all underlying blockchain technologies involved. Thus, a discussion about speed and throughput of messages is relevant. Overledger applications may require many operations on a single (or multiple) blockchain(s). In this case, the time for each (and between each) subsequent transaction needs to be properly accounted for. In the trivial case, we want to perform an over-layer action that only affects one blockchain, we want to make sure the hash of the action is stored in a valid transaction. A blockchain transaction is valid when consensus is achieved in the distributed network, meaning if a correct node appends (a set of or a block of) transaction(s) "M" to its copy of the blockchain, before appending (a set of or a block of) transaction(s) "N", then no other correct node will append "N" before "M" to its copy of the blockchain. While this concept of consensus finality is related to the idea of correctness (existence of a single global set of recognised transactions in the system), it refers to the ability of the distributed system to be resilient to the same correct transaction made multiple times (this is also referred to as the problem of double spending) and forks [29]. In this respect, a blockchain can follow a randomised or probabilistic consensus (e.g., Bitcoin) or a deterministic one (all the blockchains based on Lamport Byzantine Fault Tolerance [9]). For example, in the case of Bitcoin, to be "sure" of the commitment of the transaction we use the heuristic of the 6 confirmed blocks to reduce the probability that someone can come up with a longer chain making our transaction invalid. Now let's suppose we have our action ready at a particular time, we need to wait for a latency not deterministic before our transaction is accepted and added in a block. After this period, we also need to wait a probabilistic period of time until consensus is achieved (6 blocks in the Bitcoin case).

2.2 Filtering and Ordering Layer

One of the benefits of blockchains is its capability to store all transactions (one after the other) in a distributed, replicated (collision-free, hiding and puzzle-friendly [6]), hashed data structure, in a way that can quickly detect any changes of the log data structure or any manipulation of data (tamper-evident property). The consensus mechanism embedded in the blockchain's protocol, unequivocally decides the order of valid blocks. Thus, filtering and sorting blocks allows us to trace ownership of an asset, or to calculate the balance of an account, by replaying all the transfers and transactions from the beginning of the chain [20]. These transactions and transfers are secured because of digital signatures giving the blockchain identification, authorisation and non-repudiation properties. The signature algorithm, the key length of public and private key pairs and the hash function to produce the account address from the public key are embedded in the particular blockchain protocol. The order property, together with the non-repudiation property, solve the double spending problem within the blockchain.
The challenge of building an "over-ledger" upon ledgers is to define a criterion according to which blocks can be sequentially ordered and which blockchains' addresses can be standardised. Such criterion should intelligibly order which different blocks of different blockchains must be processed first and should

translate addresses, which are unique alphanumeric identifiers, of different blockchains, into the same format.
In our reference architecture, all the information contained in all the ledgers are put together in the Filtering and Ordering Layer (a logical, not physical layer) where a set of rules filtering a small set of messages and transactions and determines their processing order, regardless of the sequential position held in their respective ledger. For each transaction, we can extract information regarding the sources, the receivers, the coins, the script, the contract and combine them to define rules. Some ledgers allow users to append arbitrary data to transactions (see Subsection 2.1.3). Users can use this space to fill these fields with messages, or, if there's not enough space, with their fingerprints. Even if these messages are meaningful in the hosting transactions, we can use the contained information to filter and order messages and transactions in different ways. Considering it's straightforward to find an order among transactions in the same ledgers, it can be challenging to do it in a multi-ledger environment. In the previous section we focused on how to build this message and what we can achieve with them. In this section we'll introduce the ordering problem and focus on:

- How to set an order binary relation for blocks of different blockchains; and
- How to allow cross-ledger transactions by mapping addresses of the various blockchains into one address, valid in the Overledger domain.

To order the transactions within an application, the application scans the ledgers involved and places transaction hashes compliant to the Applications Blockchain Programming Interface (BPI) into a Virtual Block, called a Verification Block. A hash pointer to the Verification Block is then written to the blockchains forming part of the application. After a given interval, dependent on variables, such as number of application transactions and block height, etc, the application then re-scans the blockchains involved back to the Verification Block. Any new BPI compliant transactions appended to the blockchains that are part of the application are then appended to the Verification Block, which is updated on the blockchains involved.
Note, there is no order among transactions in the same block or amongst invalid transactions. By design, the Verification Block is keeping the order across all chains involved in the application of hashes appended after consensus has been reached.

2.2.1 Filtering Criterion in Overledger

The Messaging Layer contains potentially infinite combinations of messages and information existing in all the ledgers. To find a way to bound this set is to introduce filtering rules to drop all the messages we're not interested handling in Overledger. All these rules are related to the transactions and their fields. However, some of the fields can be filled with arbitrary data we used for the out of chain messages. This means we can set rules on these messages because their fingerprints are in the ledgers. We've two levels of where to apply these rules:

- Transaction validation rules; and
- Out of chain messages validation rules.

The first category can be a set of rules using information in the transaction. These set of rules include the choice of permitted ledgers to produce messages. Once we've picked the ledgers to use, we need to decide which fields are selected. This is difficult, because each ledger has its own transaction structure,

but we need to select fields to build consistent messages among the ledgers. Even if we chose the set of shared fields, it's difficult to create compliant messages among ledgers. For example, if we select two ledgers having a similar transaction structure and we only look to the addresses (source and receiver), we'll need to find a way to map the addresses to allow for an exchange among ledgers. For this reason, out of band messages are useful because they provide the ability to implement this map and eventually have more detailed information. In fact, we can map two addresses into two different transaction ledgers' external messages. These messages can contain a signature of a user and their fingerprints in the transactions. We can also apply complex logic to these messages, for example by utilising validation schema. However, transaction information is essential because they can contain low-level information about addresses and coin information. For example, an application can allow some messages only if they pay one of its lower level (transaction ledger) addresses, a certain amount of coin, and then use the source of that transaction to send the response. Possibly, we can also use script-smart contract information to allow a legacy application to work.

2.2.2 Ordering Criterion for Overledger

Order binary relations. To address the order problem faced in the Filtering and Ordering Layer, we resort to the Order Theory: the branch of mathematics that studies the order using binary relations. This theory provides formal tools to compare two objects within a set and to decide which one is greater than the other. Consider there are objects comparable under many different ways. For example, it's possible to sort the set of natural numbers with their magnitude or with the lexicographical order. Complex objects, composed of various fields, can be compared by using the combination of their fields. We therefore use the concept of a partially ordered set (poset) [4] to order blocks and transactions. A poset formalises and generalises the intuitive concept of an ordering, sequencing, or arrangement of the elements of a set. A poset consists of a set, together with a binary relation, indicating for certain pairs of elements in the set, one of the elements precedes the other in the ordering (i.e., not every pair of elements need be comparable). Given a set P and a relation ≤ on P elements, then ≤ is a partial order iff for all x, y , z in P we can ensure these three properties:

- Reflexivity: $x \leq x$;
- Antisymmetry: if $x \leq y$ and $y \leq x$ then $x = y$; and
- Transitivity: if $x \leq y$ and $y \leq z$ then $x \leq z$.

These orders are also known as a poset. If we can prove only reflexivity and transitivity we can call the order a preorder. If the relation has the antisymmetry and transitivity properties and instead of the reflexivity property has the irreflexivity property:

- Irreflexivity: not $x < x$

we define this relation as a strict partial order relation. If we can prove the previous three properties and the following one:

- Totality: $x \leq y$ or $y \leq x$

we can define the relationship as a total order and the set of a total ordered set, or linear orders, or chains. In poset, unique elements exist, like the least element $l \leq x$, for all x in the set and the greatest element $x \leq g$, for all x in the set. In partial order, there can be items having no elements greater/less than them and are not comparable to each other. In this case, these elements are called, respectively, minimal

and maximal. There are elements in the set that are special, concerning a subset of the order. Given a poset P, a binary relation ≤, and one of its subsets S, a lower bound of S is an element l, such that, it is less or equal to all elements s in S. Conversely, with the same assumptions an upper bound is an element u, such that all elements s in S are less than or equal to u. Another important concept is the infimum and the supremum. The infimum is a lower bound called inf of a subset S of a poset P, in which is defined the relation ≤ inf for all lower bounds l of S in P, l ≤ inf. Hence, the supremum is an upper bound called sup of subset S of a poset P, in which is defined the relation ≤, if for all upper bounds u of S in P, sup ≤ u. If the infimum is contained in the subset S, the infimum is called the Least element, in the same way, if the supremum is contained in the subset S, it's called the Greatest element. If an ordered set S has the property of every subset of S that is non-empty and it has an upper bound, it also has the least upper bound, then it has the least-upper-bound property, also known as Dedekind completeness [21]. If a total order has a least element for all its subsets not empty, the order is called well-order.
An important proposition of the set theory on which we build our ordering criterion is Zorn's Lemma [22]
ZORN's LEMMA: Given a poset P with the property that every chain contained in P has an upper bound in P; then the set P has at least one maximal element.
Moreover, we also resort to the principle of duality [26], according to which, if a given statement is valid for all partially ordered sets, then its dual statement, obtained by inverting the direction of all order relations and by dualising all order theoretic definitions involved, is also valid for all partially ordered sets. Namely, every poset P gives rise to a dual (or opposite) partially ordered set Pop. This dual order Pop is defined to be the set with the inverse order, i.e. x ≤ y holds in Pop iff y ≤ x holds in P. For this principle, the previous definitions of the least element or the minimal set can be obtained by the greatest element, or the maximal, by inverting the ordering function. Starting from a given order, this operation allows us to build new types of order.

Representation of a poset. There are alternative manners to graphically represent a poset. The first method is via Hasse diagrams, we now define. The Hasse diagram of a partially ordered set P is the (directed) graph whose vertices are the elements of P and whose edges are the pairs (x, y) for which y covers x. It's usually drawn so elements are placed higher than the elements they cover. And "y covers x" means [x, y] ={x, y}. That is, no elements of the poset lie strictly between x and y (and x ≠ y). According to the Hasse diagram, a binary relation is represented by an edge between two vertices representing two elements, such that the vertex that's below is the predecessor of the relation, and the one above is the successor of the other vertex. Orders are represented bottom-up. As an example, Fig. 10 shows the Hasse diagram of the poset P(∗ , S) where S is 1, 2, 3, 5, 6, 10, 15, 30 the divisors of 30 and is the divisibility relation that exists if one element is an integer divisor of another. For example 1530, because 30 divided by 15 has no reminder, therefore, there's an edge connecting these two elements, and 15 is below 30 because the binary relation is 15 ∗ 30. The topmost element is 30, because in the set there are no elements for which 30 is a divisor. This is a way to visualise the existence of maximals. It's not possible to spot the existence of greatest elements; it's possible there is a vertex with a predecessor and it's not connected to the presumed greatest element or any of its predecessor. In the same way, with this representation, it's easy to spot minimals, but not that easy to understand if there's the Least element.

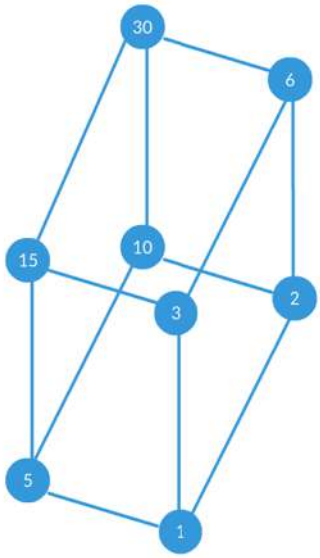


Fig. 10: Hasse Diagram of all divisors of 30, ordered by divisibility.

Another way to represent a poset is via a directed acyclic graph (DAG). In this case, each vertex represents an element, and the edges point from the predecessor to the successor. As an example, Fig. 11 depicts the order by divisibility of all the divisors of 30. DAGs are data structures easier to implement, and can be used to check properties, or find particular elements. This representation offers a way to find the set of minimals and maximals in O(n), where n is the number of vertices, by checking node by node if the list of outgoing edges is empty (maximals), or if the list of ingoing edges is empty (minimal). If one of these sets has only one element, it's a sufficient way to prove the existence of the greatest element or the least element. It's also possible to check if an element is a predecessor or a successor of another one by checking if a path exists connecting the two elements in O(V+E) with a Depth First Search [28].

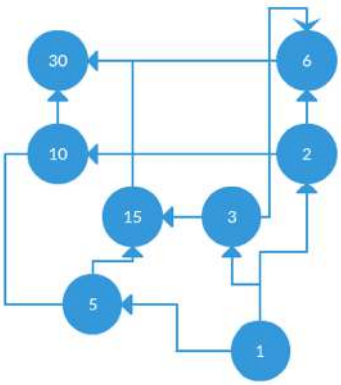


Fig. 11: DAG of all divisors of 30, ordered by divisibility.

Blocks order in popular Blockchains. Blockchain Consensus relates to the set of rules and procedures allowing us to maintain and update the ledger and to guarantee the trustworthiness of the records in the ledger, i.e., their reliability, authenticity and accuracy [29]. Those rules and procedures are to be jointly considered when designing an active network consensus validation process, because not only their individual configuration, but also their combination, determines when and how the overall blockchain agreement is achieved, and the ledger's updated with new information. From a business logic perspective, the correct order of information is critical because it can establish ownership and therefore

rights and obligations [2]. In particular, new information can be added, ordered and stored in the ledger under different rules of Consensus Immutability and Failure Tolerance [29]. The most common rules include the Proof-of-Work (PoW), the Proof-of-Stake (PoS), or hybrid solutions like "PoW and PoS" where PoW blocks act as checkpoints containing no transactions, but anchor both to each other and to the PoS chain. For example, Bitcoin and Ethereum use a set of rules respectively called HashCash [31] and Etash, based on PoW. In the PoW setup, miners connected to the network perform the task of validating the transactions proposed for addition to the blockchain by solving the inversion of a cryptographic function whose solution can only be found by brute force. In PoW the probability of mining a new block depends on the instantaneous computational power devoted to the task by all miners connected to the network under the rule "one CPU one vote". In PoW, miners are continuously asked to brute-force a one-way hash function by computing new hash values based on the combination of the previous hash values contained in the message, the new transaction block and a nonce, such that the new hash value will start with a given number of zeros ≤ target. In this way it's easy to build a strict binary relation among blocks.
Differently from PoW, in PoS the creator of a new block is selected in a deterministic way, depending on its stake. Block generation is linked to the proof of ownership of a certain amount of digital assets (e.g., digital currencies) linked to the blockchain. The probability a prover is selected to verify the next block is larger, the larger the share of assets this prover has within the system. The underlying assumption is users with a large share of the system's wealth are more likely to provide trustworthy information with respect to the verification process, and are therefore to be considered a trusted validator [32].
In the case of DLTs where the storage of data isn't based on chains of blocks, such as Ripple [31], the addition of new information on the ledger is based on iterative processes. Each node builds a transaction set based on the newer transactions it knows (candidate set) and broadcasts it to its trusted network, as a proposed transaction set to be applied to the ledger. When a node receives a proposed transaction set by trusted nodes, it implicitly votes each transaction into the set with "yes" or "no" by including it in a newer proposed transaction set. If a certain percentage of nodes in the trusted network of a specific node votes for a transaction, then that node will add it to its transaction set. After awhile, the percentage threshold rises by making the transaction set converge into one, which is added to the ledger concerning a specific timing scheme.

Blocks and transaction order in Overledger. Among other things, we have seen that consensus techniques order the blocks. It's therefore convenient to represent, with a Hasse diagram the order of blocks B(< , S) where: S is the set of all blocks, < a relation that states which block comes first. B is a total strict order, meaning given two blocks, it's always possible to choose which comes first. As an example, Fig.12 represents the Hasse diagram of an order problem with 4 blocks.

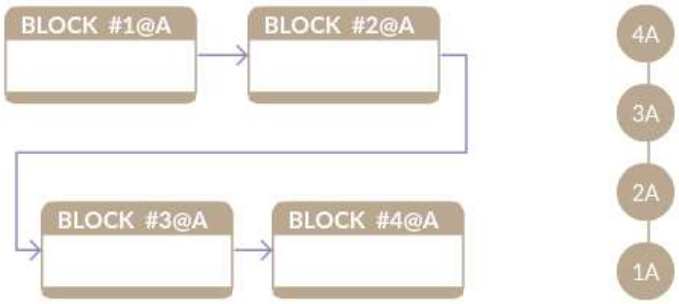


Fig.12: Hasse diagram of the order B(<, S), where S is the set of blocks{1A, 2A, 3A, 4A}.

If two blocks have two conflicting transactions, the transaction in the greatest (latest) block isn't valid because that coin's considered already spent. This is how ordering partially solves double-spending. One challenge is what happens if the two conflicting transactions are present in the same block. Since transactions within the same block have the same timestamp, we can model the set of transactions as a total poset T(≤ , W), where W is the set of all transactions contained in the blocks, and ≤ is the binary relation that compares transaction order. As an example, Fig.13 represents the Hasse diagram of a poset problem with 3 blocks and 5 transactions. Luckily, this problem is automatically solved in many blockchain protocols, because blocks with conflicting or invalid transactions are not eligible to be added, and when they're proposed as the next block in the ledger, they're dropped. In this way, it's not possible to have conflicting transactions either in different blocks or in the same block [29].

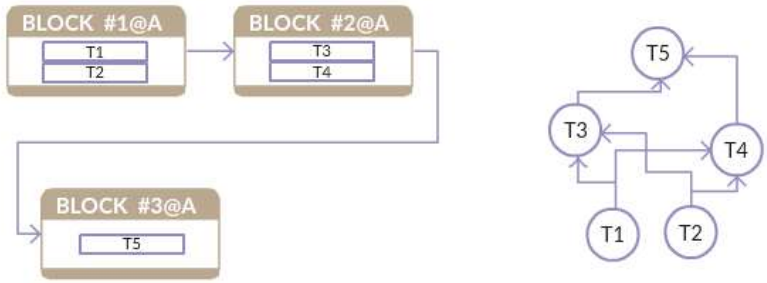


Fig 13: Hasse diagram of the poset T(<,W), where W is the set of transaction{T1, T2, T3, T4, T5}.

Overledger block order. While the order set of blocks and transactions within a particular blockchain can be modelled as total order, the comparison of the order of blocks or transactions across different blockchains is impossible, as those elements exist inside different domains.

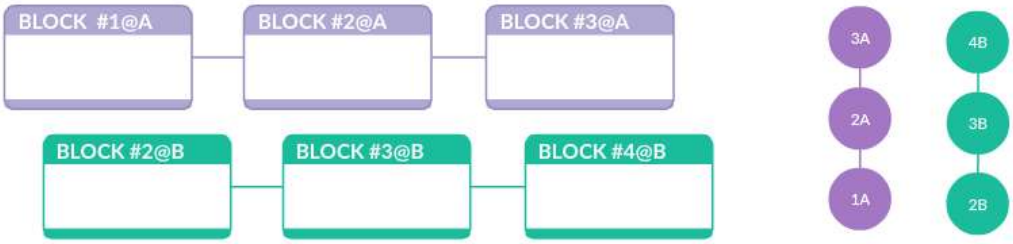


Fig. 14: Blockchains and overledger representation.

This problem is represented in Fig. 14 showing the Hasse diagram of the set of some blocks of two different blockchains: blockchain A and B. The order is the poset O(≤ , S), where S is the set{1A, 2A, 3A, 2B, 3B, 4B}and ≤ is the binary relation deciding which block comes first. There's no way to compare block 1A with 2B, or any other couple of blocks belonging to different chains. The easiest solution to this problem is to use timestamps to decide which block comes first. However, by dealing with different consensus mechanisms, the timing scheme regulating the way new blocks are added to the chains, can make this choice non-compliant with cross-ledger business logic. Moreover, it's possible some chains don't use a timestamp comparable with others. In fact, timestamps define a time range in which the transactions are considered to have occurred. Timestamps are expressed as windows, because in a distributed system there is no one "true" time. For example, in the Bitcoin blockchain a block is rejected if it contains a timestamp: 1) lower than (or equal to) the median timestamp of the previous eleven blocks; and 2) greater than (or equal to) the "network-adjusted time" + 2 hours. Nevertheless, this isn't a problem

in Overledger because there's no need to compare transactions moving assets existing only in their blockchain domain, and therefore we don't have conflicting transactions. The situation changes when we'll introduce cross-ledger transactions (in Subsection 2.3). In this case, we'll explain how we can compare some of the blocks of different chains and detect conflicting transactions.

Cross-ledger transaction and block poset. Fig. 15 provides an example of a cross-ledger transaction. There are two blockchains: A "purple" and B "green". Each block contains a set of transactions identified by the unique field serial and composed by a 'Coin' field indicating the asset moved in the transaction, and a 'Public key' field, showing which user's moving the asset to whom. In block 2 of blockchain A (#2@A) the transaction indexed 2 is a cross-ledger transaction that moves the asset 3 of blockchain A (3@A) from the user U4 in blockchain A (U4@A) to the user U1 of blockchain B (U1@B).

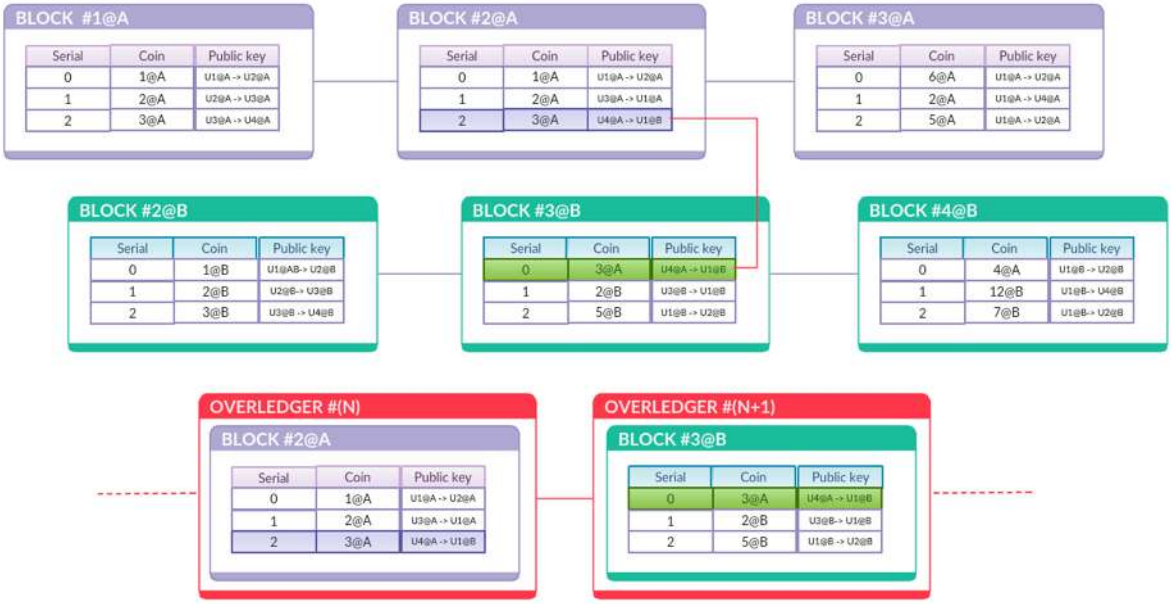


Fig.15: Cross-ledger and Overledger transaction representation.

The same transaction appears according to blockchain B semantics into block (#3@B) at index 0. For the sake of simplicity, we assume the blockchain semantics are the same. These two transactions represent the same transfer. Therefore, it's evident the two blocks containing these two transactions happen together, concerning the business logic in our example. However, since these two blocks can be added with different consensus methods, it's very unlikely they're added at the same time. Moreover, to avoid inconsistency, it's mandatory to find a way to allow atomic cross-ledger transactions in the unfortunate case when only one of the two blockchains record the transaction. In this Overledger representation, even if the transactions of these two blocks happen at the same time, regarding our business logic, they're added into subsequent over blocks. This choice will be explained later when we'll introduce a protocol to achieve atomicity, as the user owning the asset proposes the first transfer (see Subsection 2.3). In Fig 16 there's the Hasse diagram of the set of all blocks in the different blockchains, ordered by the binary relation, stating which block comes first according to the Overledger business logic. While in Fig. 14 it wasn't possible to compare blocks in different blockchains, in the case of Fig. 15, we're able to compare them because of the cross-ledger transaction. Namely, we can say that (#1A ≤ #4B) or (#2B≤#3A). Thus, we can model the Overledger as a chain containing blocks of the different chains. A block of the Overledger (over block) can include one or more blocks of different blockchains.

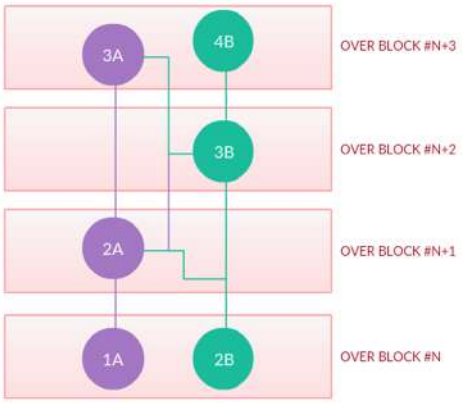


Fig. 16: Hasse diagram of Poset O(≤, S) where S are the blocks in the blockchains in Fig. 15.

This poset has a set of minimal blocks for every block because of Zorn's lemma and duality. Fig. 17 represents Overledger based on the cross-ledger transactions shown in Fig. 16.

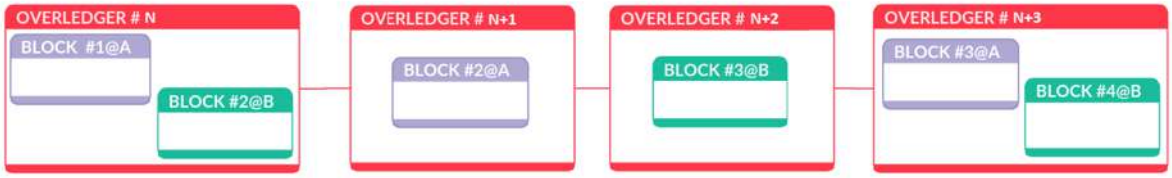


Fig. 17: Overledger modelled as a blockchain of block set 4.6. Note that the block 2 and 3 can be grouped together.

Each Overledger block can contain many other blocks of different blockchains. Therefore, it includes the set of all transactions in those blocks. In the example shown in Fig. 15 we can model the Overledger block as a regular block with all the transactions, because we're sure there's no conflicts. Therefore, we can surmise they happen at the same time.

Fig. 18: Transactions of the #N block of Overledger shown in Fig. 17.

Blockchain of Blockchains. If there are more blocks of the same blockchain, we can't model all the transactions, since they were happening the same time as this, it may lead us to enter in the same set of conflicting transactions. In fact, a strict order relation exists between transactions in different blocks of the same blockchain. For this reason we consider them all equal.

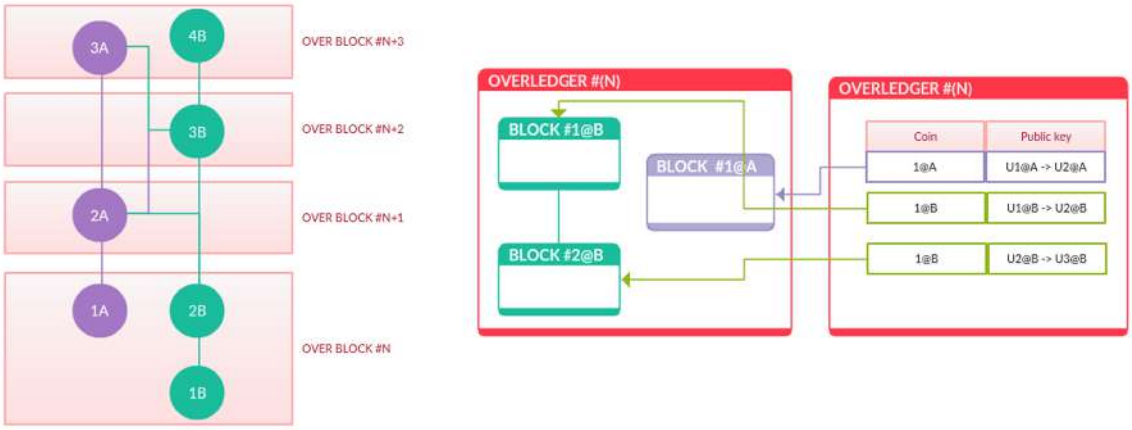


Fig. 19: Conflicting transactions in the same over block.

Fig. 19 is a Hasse Diagram showing the order of the blocks which shows a cross-ledger transaction between the blocks #2A and #3B. In this case, the transaction in block #1B transfers the coin 1@B from U1@B to U2@B; the transaction in block #2B moves the same coin from U2@B to U3@B. Now there's a clear, strict order relation between these two transactions. We can't ignore this order and consider these transactions happened at the same moment, because the transfer that moves the coin from U2@B isn't even valid, since U2@B doesn't own the coin yet. Hence, Overledger treats the blocks, or transactions set, as a poset rather than a regular set. It leads the block in our model to contain a little piece of different chains. We obtain what in computer science is known as a list of lists, in this case, a blockchain of blockchains.

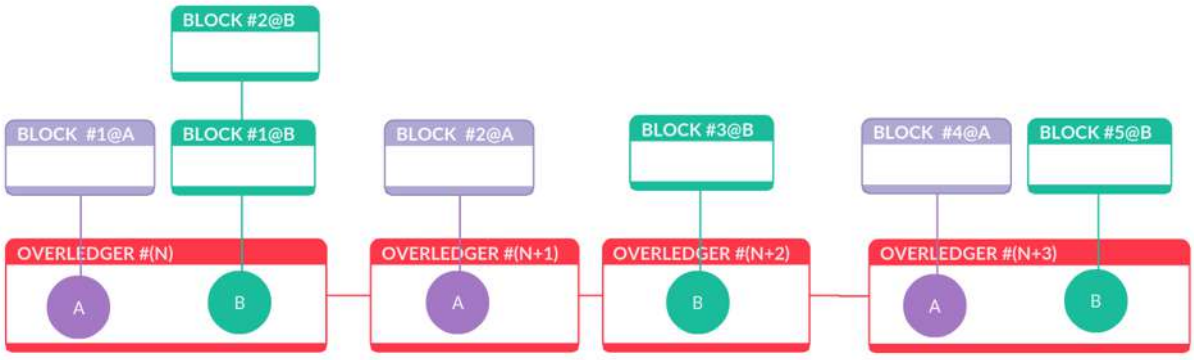


Fig. 20: Blockchain of blockchains representation of Overledger whose Hasse Diagram is in Fig.19.

Fork Issue on multi-ledger applications. We described blockchains as a list of valid blocks and we said there are common rules to accept whether a new block is valid or not. However, in some cases, a blockchain can suffer the fork problem [3]. A fork is a situation in which a blockchain splits into two (or more) separate chains temporarily or permanently. Forks naturally occur when miners unintentionally solve the PoW puzzle at nearly the same time and generate distinct blocks that succeed the same block. This is

an undesirable situation in which transactions in the two (or more) branches are not ordered and thus could be conflicting. The rule in such situations is the longer branch is the main chain (long-chain rule). After a fork forms, one branch quickly becomes longer, and is extended by all miners, thus the system's state is agreed upon. Blocks in the shorter branch are pruned and their transactions are ignored, as if they never happened. However, forks can also occur as a consequence of the use of two distinct sets of rules trying to govern the same blockchain. In this case, the fork is permanent and can be either soft or hard [33].

Regardless of the reason, forks can compromise the blockchain business logic. For example, a branch containing a set of messages and another one containing a completely different set of messages, or with a different order. This problem is solved in single-ledger environments with other techniques, unfortunately not applicable in our context. In a fork situation, even if the branches are in conflict with each other, they're consistently alone. Therefore, a way to solve this problem in a single-ledger environment is to have a clear and universally recognised set of rules (like the long-chain rule), independently allowing all users to univocally pick the valid branches among the various ones.

Let's imagine an Overledger application which interacts with two distinct blockchains, A and B (Fig. 21). If a fork happens after block 3 of blockchain A (#3@A). The Overledger application may not be aware of the fork and may append some transactions on block 4 (#4@A1). If we later discover the surviving branch doesn't include block (#4@A1), the ordering, and therefore the business logic, is compromised. Even if single blockchain A is fork-resistant and can backup in a consistent state, we can't be sure the restored consistent state is the one consistent with our business logic, and with the other blockchain, B, involved in the same Overledger application.

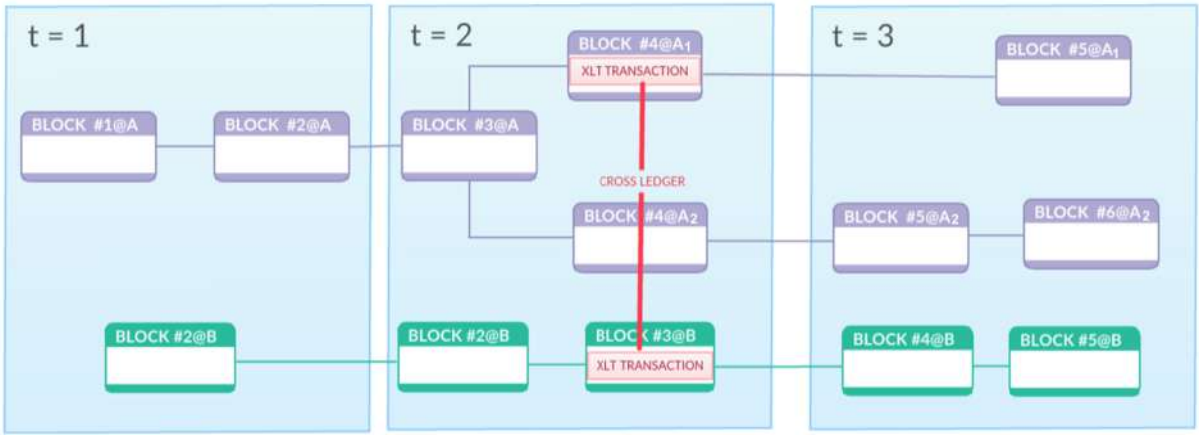


Fig. 21: Example of a fork happening in one of the ledgers of Overledger, leading to inconsistency.

2.3 Cross-Ledger Transactions

In a distributed environment, it's challenging to build applications. On the one hand, we want our applications to guarantee validity in the event of failures and errors. On the other hand, we want high availability, security and openness. It becomes harder in an environment where nodes behave according to the BAR model (Byzantine, Altruistic Rational). In this context, one should develop a system that's, at the same time, ACID compliant and Byzantine Fault tolerant (BFT). Achieving ACID and BFT in ledger logic is challenging because of failures, errors and malicious behaviour in all the sub chains and their

interactions. In the previous section we introduced cross-ledger transactions and studied their importance, to build an order between blocks in different blockchains and an order between transactions.

2.3.1 Blockchain and transaction properties

In computer science, the acronym ACID stands for Atomicity, Consistency, Isolation and Durability [34]:

- Atomicity: All changes to data are performed as if they're a single operation. That is, all changes are performed, or none of them are;
- Consistency: Data is in a consistent state when a transaction starts and when it ends;
- Isolation: The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialised; and
- Durability: After a transaction successfully completes, changes to data persist and aren't undone, even in the event of a system failure.

These properties guarantee the validity and compliance of transactions to the business logic. It's not easy to satisfy all these properties and systems requiring high availability of data and high scalability. It's better to use the BASE property of eventual consistency (Basically Available, Soft State, Eventually Consistent). The BASE acronym takes its origins from the CAP theorem [35]. The CAP theorem states that a distributed computer system cannot guarantee all of the following three properties at the same time:

- Consistency: Every time a user manages to read data he will receive the most recent value of that data, but they're not guaranteed to receive an answer;
- Availability: Every time a user will try to read data he will always receive a reply, but is not ensured he receives the most recent values; and
- Partition Tolerance: The system will work even after many messages were lost among two different partitions of the system.

A BASE system gives up on consistency in exchange for availability. Basically Available indicates the system does guarantee availability, in terms of the CAP theorem. Soft State indicates the state of the system may change over time, even without input. This is because of the eventual consistency model. Eventual Consistency indicates the system will become consistent over time, given the system doesn't receive input during that time.
BASE architecture is used in domains where there's a huge demand for data, and approximate answers can be tolerated as well as using stale data. E-commerce sites can show the last item in stock to many users, and it can happen that many users may buy the same item. Then it's not an issue for the e-commerce site to sell the item to only one customer and to apologise to the others. In contexts where users are dealing with asset transfers, there's a need to assure validity and all the ACID properties. Blockchains don't guarantee, in a deterministic way, all the ACID properties, which we consider in the next section.

2.3.2 ACID and Blockchain

Atomicity. Each transaction is not breakable into smaller operations, and its execution is binary. Either it succeeds or aborts. Blockchain always guarantees atomicity since a transaction can only be present or absent in a block. Therefore, there are no intermediate states where the operation can fail. This is also the

case for smart contracts, because even if it's true there's no precise state of those assets in the contract, the constraints ruling those assets are either committed or aborted.

Consistency. Before an operation is executed, the system is in a state compliant with the business logic. After the execution of the operation, the system is in a different state, still compliant with the business logic. The changes in states shifts the focus to assure data integrity within the system is upheld. One should be aware, consistency has a different meaning from the consistency in the CAP theorem. Blockchain guarantees consistency, in fact, every time transactions are validated a block is appended. Appending a new block should not lead the blockchain to an invalid state because the consensus mechanism assures it's a valid block that takes the blockchain to a valid newer state.

Isolation. Each transaction must be executed in a way that won't interfere with others having the same result of a system that executes all the transactions sequentially. A block's strict order in the blockchain guarantees the isolation property.

Durability. Committing a transaction produces changes that will be persistent. Blockchains with non-deterministic consensus agreement finality [29], like Bitcoin, don't guarantee this property in a deterministic way, because the probability to disagree decreases over time. However, the probability of it incurring in a fork can't be completely eliminated. Therefore, transactions within blocks are not persistent. However, blockchains with deterministic consensus agreement finality, like all those based on Lamport Byzantine Fault Tolerance [9], guarantee this property, because consensus converges with certainty and transactions are immediately confirmed/rejected in/from the ledger.

2.3.3 BASE and Blockchain

Eventual Consistency. According to this model, different nodes can store different and conflicting states of the logic. In a sufficiently long enough period, nodes detect conflicts and solve them. This means nodes will eventually give the most updated version of data. This architecture enforces liveness because nodes don't need to be sure their data version is the most update. They can directly answer the request with the best answer they can give.

Strong eventual consistency (SEC). In some contexts, data updates bring the state from one point A to the same point B, regardless of the order of transactions. In other words, if two nodes receive the same messages, but in different order, they're considered to be in the same state anyway. For the same logic, a counter that receives the same increment operations and decrement operations will be in the same state (value), regardless of the order of these operations. This property isn't true for transactions and can lead to an invalid state, because the set of transactions is a poset and we can't ignore the order, as shown in Fig. 19. However, blocks in the blockchain have the order that guarantees consistency among transactions embedded in their format. If a node receives the same blocks, but in a different order, it can rebuild the proper order by using hash pointers. This means blockchains can be considered as Strong Eventual Consistency (SEC) systems, that can be ACID (in the case of deterministic consensus), or converge in probability to ACID (in the case of non-deterministic consensus). In the last case, the probability can be estimated to be proportional to the number of confirmed blocks succeeding the block of interest.

2.3.4 Cross-Ledger Transaction and ACID

Atomicity. In cross-ledger transactions we need to record the data in all blockchains interested in the exchange of information. In the easiest case, like the one shown in Fig.15, the transaction needs to be recorded in both the domains of blockchain A and B. The transfer operation is no longer atomic, as for single-ledger operations, because it consists of two sub-operations.

Consistency. For various reasons, it can happen that a cross-ledger transaction is recorded in only one of the blockchains and hence breaking the consistency among blockchains. In fact, if for example, blockchain A records the transfer of an asset into the domain of blockchain B, according to A, the asset will be considered to be in B. Conversely, blockchain B may consider that asset to be still in A. In that case, the two blockchains don't agree, and since an asset can exist in only one domain there's a data integrity violation.

Isolation. Cross-ledger transactions don't guarantee isolation. Let's suppose on blockchain B there's a transaction that moves an asset from A to user U1@B. Suppose this transfer failed on blockchain A. The user U1@B can use his partial information to move an asset he doesn't own. This is an example of what can happen if acting upon uncommitted information.

Durability. Since cross-ledger transactions are based on blockchains, the durability property is only guaranteed for blockchains based on deterministic consensus. Instead, for blockchains based on non-deterministic consensus, one could estimate a probability that's the product of the confirmation probabilities of all blockchains.

2.3.5 Two-phase Commit and Cross-ledger Transactions

In transaction processing, databases, and computer networking, the two-phase commit protocol (2PC) is a type of an atomic commitment protocol that validates transactions in distributed environments. It's a distributed algorithm that coordinates all the processes participating in a distributed atomic transaction, on whether to commit or abort (roll back) the transaction [37]. In other terms, 2PC ensures either all the databases are updated or none of them, so that the databases remain synchronised. The algorithm is even able to achieve its goal, in many cases, of temporary system failure (involving either process, network node, communication, etc. failures). All computer nodes in the network interested in the transaction (in our example in Fig. 22, Node1 and Node2) need to express a commit or an abort to a coordinator, according to a time scheme. If the coordinator receives all commits from the nodes, it will validate the transaction, otherwise it will abort it. It's called a two-phase commit, because a commit can be reached after two phases. In the first one (voting phase) all nodes express their intention to commit or abort. In the second phase (commit phase) the coordinator will commit the transaction only if there's no abort or error messages.
The problem here with cross-ledger transactions is we need to distribute the coordinator's job. In the proposed scheme (Fig. 23) the role of the coordinator is made by the two entities (two clients, or client and server) that exchange an asset. They start the interaction posting a "ready" message that acts like a commit request on all blockchains involved. When they succeed to add a "ready" message on all the chains, they then confirm and commit the transaction on the chain where the transfer started. All messages "ready" have a hash pointer to the proposed message to prove the user's intentions. The final commit message contains all the hash pointers of the "ready" messages. This simple scheme locks the coin until a successful transfer. One could build more complicated models only including the ability to abort in some of the involved blockchain (e.g. the one where the transaction has been proposed).

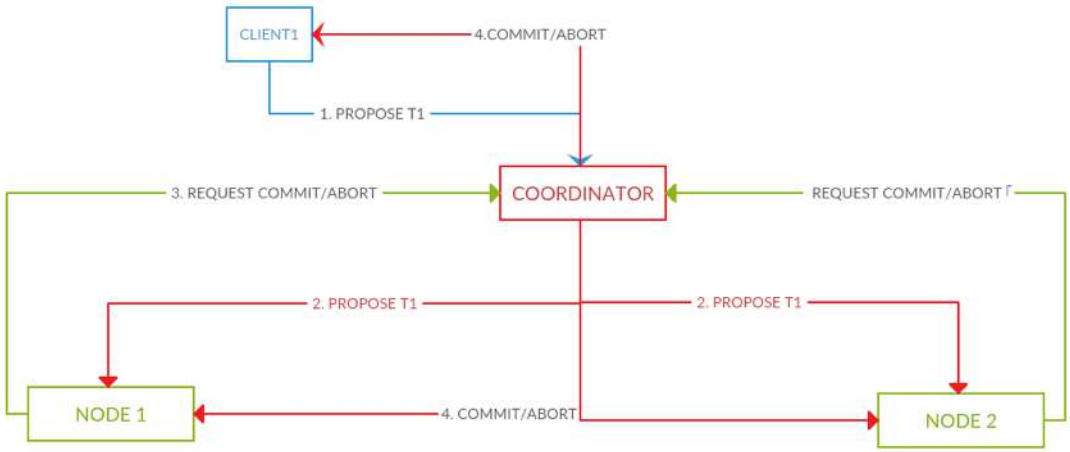


Fig. 22: Two phase commit scheme.

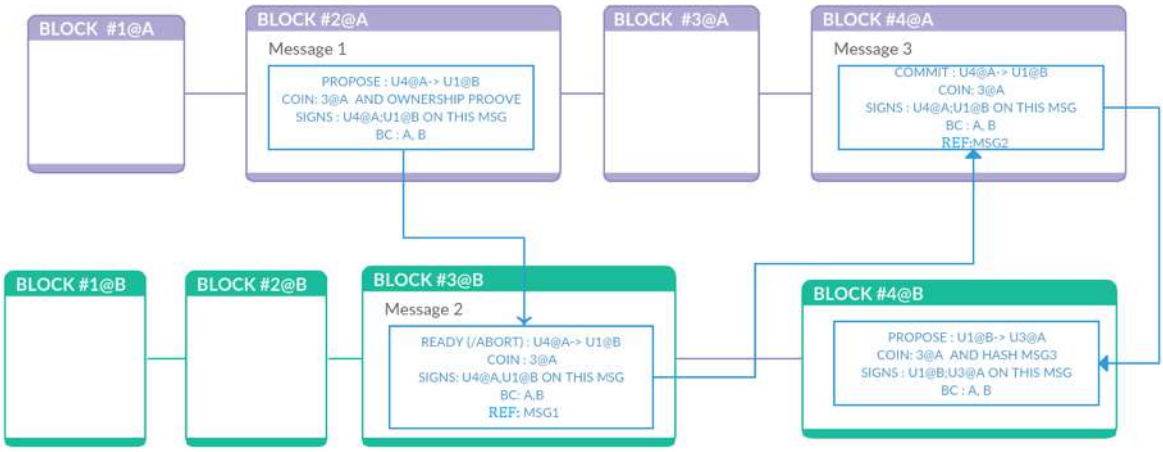
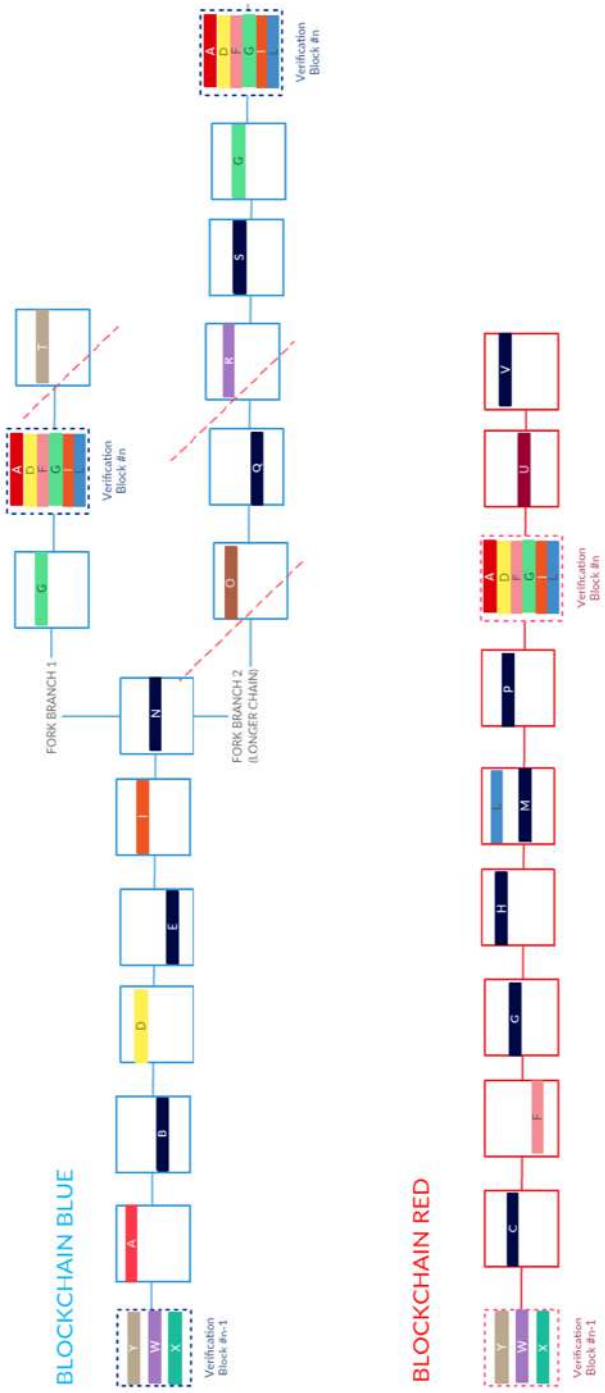


Fig. 23: Two phase commit scheme on two ledgers.

On rare occasions, the application may write a transaction to one of the blockchains involved and this transaction may be invalidated by the consensus mechanism by performing a fork. The ordering solution shown in Fig. 24 solves this issue by appending the last known Verification Block, common across the blockchains involved in the forked chain. The application then scans back across the blockchain re-appending any transactions invalidated by the fork. Both the standard ordering method, and the cross-ledger fork issue, are shown in the diagram of Fig. 24. We have named this solution as the Verification Block.

Fig. 24: Verification Block Process.



2.4 Application Layer

In this section, we'll explore the Application Layer: the upper part of our reference architecture. In this domain, there are several isolated applications that have their business logic independent from the lower component. Applications have the chance to communicate with each other by putting messages on the Messaging Layer. If these messages are compliant with the filtering rules of the other application, they can

flow through the Filtering and Ordering Layer to the Application Layer. Applications also have the chance to implement communication mechanisms not using this scheme. This scheme allows anonymous users to send a message to the application if it's compliant with its business logic.

2.4.1 Overledger Applications

In the previous sections, we explored from a theoretic point of view, how to build an Overledger architecture by adding messages as meta information on different ledgers, giving them an order and making them part of a more complex application business logic. In this section we'll describe the steps to perform a basic transaction, i.e. append a message in Overledger. We can start by providing a clear and formal description of what Overledger is, what its building blocks are, and how it can be used via the Blockchain Programming Interface (BPI).

Overledger definition. Overledger can be described as a sorted list of messages which satisfy a set of unambiguous properties. A set of properties define the valid format of a message, how to build the fingerprint and other requirements dependant on the methods required. A transaction must contain the fingerprint of a valid message to be considered part of an Overledger application. The list of these messages defines what we call the Messaging Layer of Overledger. Another set of rules determines how to sort the valid messages in a sorted list we called the Ordering and Filtering Layer. A system using Overledger reacts to this sequence and can change its state. Any change in one of these sets of rules results in a different list of messages, or a different permutation. More systems, with different control logic, can share the same Overledger if they respect the same rules.

Blockchain Programming Interface (BPI). Multi-ledger applications wanting to use Overledger will need to define two sets of (mandatory and optional) rules. These rules determine how a wire protocol will interact with the Overledger system and with other users/applications. A non-exhaustive list of rules includes:

- Accept messages that can be validated by specific schema;
- Accept messages only if they respect certain pattern sequences (e.g. in a 2PC we want to receive the "propose" message before the "ready" message);
- Accept messages only if their fingerprint (hash) has been appended on a particular set of ledgers;
- Accept messages only if their fingerprint has specific source and recipe addresses; and
- Accept messages only if their fingerprint is spending at least a certain amount of cryptocurrency.

An application can define more complex messages, it can use any combination of them (e.g. different messages can have different validation schema, different rules on the transactions hosting their fingerprint). The rules can also involve other information, for example, the script or the contract they contain. In this respect, this schema is similar to the one used by systems which expose APIs. In fact, those systems, at the application level, publish a file and a library to interact with the system as it is an Abstract Data Type (ADT). Likewise, our set of rules describe how to interact with Overledger. We assume the approach is the same, having the API on the Application Layer and rules on the Messaging Layer, and are technology independent. We call our approach, BPI (Blockchain programming interface). A BPI is essentially a marked-up text that defines the specific Overledger rules an application must follow in order to add messages, or to read the output of a write, to the ledgers in question. The BPI should be automatically read by the clients that implement the required methods according to their technology. We're now creating an Overledger Software Development Kit (SDK): a set of software development tools

that will allow applications to define the rules (BPI) without the need to address the low-level details regarding transactions and messages. Through the SDK, we'll also enable additional methods, where appropriate, to interface with other lower level functions of the ledgers in question.

After the final overview of Overledger, and the introduction to the Overledger BPI, we provide a simple example of a typical transaction journey. We describe the steps to be followed in order to append a valid message on Overledger. In this example, if a user (client) wants to append a specific message, it should follow the following rules:

1. The client reads the requirements for the message it wants to append;
2. The client builds a valid message M1 and calculates its fingerprint H1;
3. The client builds a valid transaction T1(inserting the fingerprint) and proposes it to the systems in charge of proposing transactions;
4. T1 is appended on a ledger accepted by the BPI; and
5. The client sends the complete message to the application and the transaction T1 that contains its fingerprint.

Note that the rules in this example are interchangeable.

2.4.2 Application level responsibility

In Sections 2.1 and 2.2 we discussed the different roles of the Messaging Layer and the Filtering and Ordering Layer. We explained those are logical layers, but we didn't explain who's in charge of implementing those features. Filtering messages requires a deep knowledge of all the different blockchains included in the Overledger application. For this reason, it would be convenient to build open source libraries helping the application developer to use a high-level function to interact with blockchains. Let's take an example of a write operation for a client sending a message to a particular application (app_id=1):

```
import bitcoin as btc
import ethereum as eth
import json
import hashlib
import requests

# Message definition#
jstring = {'app_id'=1, message'='Hello world', 'client'=1}
msg = json.dumps(jstring, sort_keys=True)

hash_function = hashlib.sha256()
hash_function.update(msg)
hash_message = hash_function.digest()

r = requests.post(server_address, json=msg)
```

```
eth.add_msg_out_of_the_chain(hash_message)
btc.add_msg_out_of_the_chain(hash_message)
btc.add_msg_out_of_the_chain(hash_message, mode='OP_RETURN')
```

In this example, the code takes all the responsibilities for all layers. The first lines are both related to messaging and filtering: the definition of a particular message format, the values of the particular message instance, the chosen hash function and the method of how the message is added to the ledger. The last two lines of code show the Bitcoin chain can allow a different way to add information to its ledger. In the last line, the optional parameter mode specifies which one must be used rather than the default option. This performs the post-operation send of the entire message through the network. OP_RETURN is a script opcode used to mark a Bitcoin transaction output as invalid. Since any outputs with OP_RETURN are provably un-spendable, OP_RETURN can be used for digital asset proof-of-ownership and other transaction-based business logics.
The server is responsible for checking if the message is valid and, in this case, if the hash of that message is on the blockchains of interest, with the proper transaction parameters (addresses and coins). To check if the message is in the chain, the application needs to perform a scan of the newer blocks and react, when it finds the message. This can lead to solutions that need a timeout and complex protocol. If the server finds the hash message, it can react, because in the JSON file there's all the information of the receiver. In this case, there's the field "client" in the message equal to one. That number can refer to a client address or to a set of configurations, to allow the server to reply to it with this schema, adding messages on blockchains. Another design solution can assign the responsibility to check the hash to the client, that can scan the interested ledgers, until it finds the transaction and can prove by providing the transaction's hash pointer (e.g. Merkle tree proof) to the server. In this case, the server can quickly check the lowest details and doesn't need to pull all the interested chains to scan.
In this last example, we introduce, in the import statement, a library "quant", that takes the responsibility for tasks of the Filtering and Order Layer. The application loads the configuration file that can be hosted on the local storage, or can be downloaded every time, to guarantee consistency between the application upgrade and the client application. In this case, the developer needs to only focus on the business logic of the client.

```
import quant.app_builder as q

app = new q.client_application()
#in conf.txt there are all the information
#about the Filtering Layer
app.set_configuration(./conf.txt)
jstring = {'app_id'=1, message'='Hello world'}
msg = json.dumps(jstring)
app.sendmessage(msg)
```

3. Use Cases

There are many uses for Overledger produced applications, all of which are far too numerous to list in this paper, but as an example, we've included one of the simplest.

3.1 CQRS-ES and Overledger

Command Query Responsibility Segregation. CQRS (Command Query Responsibility Segregation) is an architecture pattern introduced by Greg Young [27]. This pattern uses different models for writing and reading the model. The writing model should manage the command operations that, following the CQS (Command Query Separation) vocabulary, can update the model and must not return a value. The reading operations should handle the query operations that mustn't change the system state (no side-effects). In some domains this pattern can be useful for breaking down the complexity of a system, especially where reading operations and writing operations have different time scales.

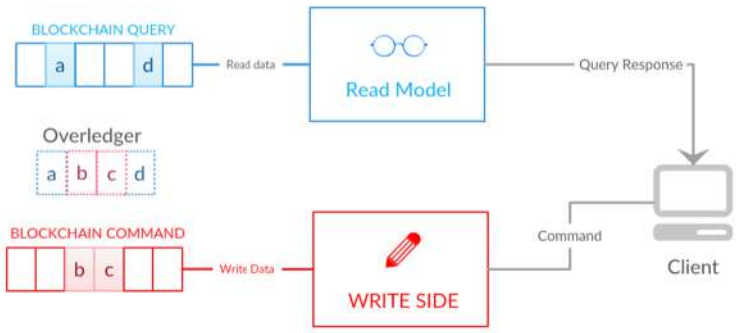


Fig. 25: CQRS Architecture Schema for Overledger.

Overledger can be a good fit for this pattern implementation (see Fig. 25). We can use two or more different blockchains, some for appending commands and others for querying the status. This can be useful, not only because the application may have a different scale for reading and writing, but because different blockchains have different levels of security and different throughput. For example, it could make sense to use a faster blockchain to collect the commands, and a more secure (and probably lower speed) blockchain to store the valid transaction in the application logic; or use a different blockchain to show different types of aggregation according to the client's needs (this blockchain could only store the client state or Verification Blocks).
Overledger Event Sourcing (OES) is a pattern that builds the status of the system scanning all events that have changed it. When a new event happens, rather than updating the state of the system, events sourcing architecture appends it to the event stream. The state is calculating by replaying all the events. Since appending events is a single operation, OES is inherently atomic. The way the state is calculated has

many similarities with blockchain technology. In Fig. 26 events are appended to one blockchain and different domains of the applications retrieve those event data for different reasons. For example, one domain can be interested in storing data for ETL, one for optimising searches, and another for processing the event and streaming the output to another blockchain.

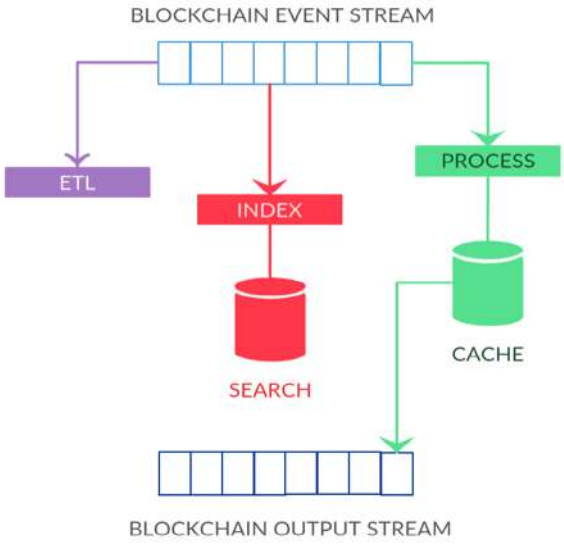


Fig. 26: CQRS Architecture Schema for Overledger.

The similarities between this pattern and the mode of operation of the blockchain makes its integration with Overledger natural. In the literature CQRS and Event Sourcing (ES) are often combined and used together (CQRS-ES), noting this is still applicable with Overledger.

Communications. While email, in its current state, is a very effective medium, it does, however, have one flaw. The receiver of the email can refute the delivery of the communication. Even where controls, such as delivery/read receipts, are requested, these can be ignored by the recipient. Now if we were to use Overledger as a simple messaging solution we could produce the process flow in Figure 27.

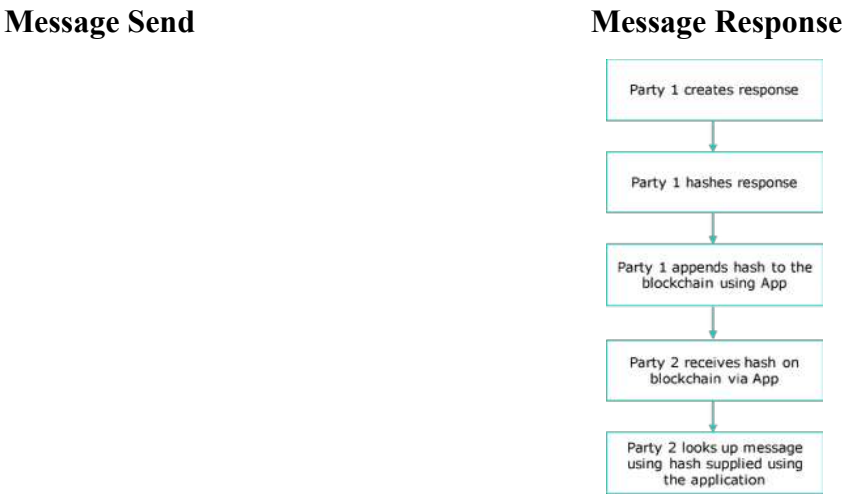


Fig. 27: Simple Overledger Message Flow.

4. Conclusion

In the above paper, we propose the design of a new operating system able to increase interoperability between different DLT technologies. The Overledger operating system proposes an approach both to order and to manage the blockchain blocks forming part of the Overledger application. It does this so as to eliminate double spending problems, while maintaining the ability to react to situations where the DLT may become subject to a fork, as a result of not being part of the accepted consensus. We solve the above challenges by introducing both the BPI (Blockchain Programming Interface), and the Verification Block, as described above. These solutions, when coupled with a 2-Phase-Commit, can facilitate cross-ledger transactions, both at a transactional and a Messaging Layer.
Finally, the paper describes that by leveraging features within the underlying functionality of the DLT's, these can be utilised to impose a CQRS architecture pattern. This can then be held and managed externally to interface not only across DLTs, but also to integrate with legacy systems and external data sources.

Bibliography

1. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
2. Tomaso Aste, Paolo Tasca, and Tiziana Di Matteo. "Blockchain Technologies: The Foreseeable Impact on Society and Industry." Computer 50.9 (2017): 18-28.
3. Paolo Tasca, Digital Currencies: Principles, Trends, Opportunities, and Risks (September 7, 2015). Available at SSRN: https://ssrn.com/abstract=2657598
4. Birkhoff, Garrett. Lattice Theory. 25 (3rd Revised ed.). American Mathematical Society. 1940.
5. Luke Dashjr Mark Friedenbach Gregory Maxwell Andrew Miller Andrew Poelstra Jorge Timn Adam Back, Matt Corallo and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. 2012.
6. Edward Felten Andrew Miller Steven Goldfeder Arvind Narayanan, Joseph Bonneau. Bitcoin and cryptocurrency technologies. 2015.
7. Ryan Shea Michael J. Freedman Princeton University Jude Nelson, Muneeb Ali and Blockstack Labs. Extending existing blockchains with virtualchain. 2017.
8. Jae Kwon and Ethan Buchman. Cosmos. a network of distributed ledgers.
9. Marshall Pease Leslie Lamport, Robert Shostak. The byzantine generals problem. 1982.
10. Jude Nelson Muneeb Ali, Ryan Shea and Michael J. Freedman. Blockstack: A new internet for decentralised applications. 2017.
11. Barbara Liskov Miguel Castro. Practical byzantine fault tolerance,laboratory for computer science, massachusetts institute of technology. 1999.
12. Pierre Karpman Ange Albertini Marc Stevens, Elie Bursztein and Yarik Markov. Aion: The third-generation blockchain network. 2017.
13. Pierre Karpman Ange Albertini Marc Stevens, Elie Bursztein and Yarik Markov. The first collision for full sha-1. 2017.
14. The Digital Asset Platform. Non-technical white paper. 2016.
Manny Trillo.
15. Stefan Thomas and Evan Schwartz. A protocol for interledger payments. 2016.
16. Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. 2016.
17. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Yellow Paper.
18. Jae Kwon. Tendermint: Consensus without mining. 2014.
19. Matthew, Spoke and Nuco Engineering Team, Aion: The third-generation blockchain network, 2017
20. Paolo Tasca, Shaowen Liu, and Adam Hayes. The evolution of the bitcoin economy: extracting and analyzing the network of payment relationships. Forthcoming Journal of Risk Finance, 2018.
21. Aliprantis, Charalambos D; Burkinshaw, Owen. Principles of real analysis (Third ed.). Academic. 1998.
22. Campbell PJ. The origin of "Zorn's Lemma". Historia Mathematica. 1978 Feb 1;5(1):77-89.
23. Bitcoin Wiki. 2018. OP_RETURN. [ONLINE] Available at https://en.bitcoin.it/wiki/OP_RETURN. [Accessed 13 January 2018].
24. Ethereum Github. 2018. Ethereum Development Tutorial. [ONLINE] Available at https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial. [Accessed 29 January 2018]
25. Ripple. 2018. Transactions Format. [ONLINE] Available at https://ripple.com/build/transactions/. [Accessed 29 January 2018].
26. Davey, B.A.; Priestley, H. A., Introduction to Lattices and Order (2nd ed.), Cambridge University Press, 2002.
27. CQRS. 2018. Welcome to CQRS Info. [ONLINE] Available at https://cqrs.wordpress.com/2010/09/08/welcome-to-cqrs-info/. [Accessed 29 January 2018]
28. Even, Shimon, Graph Algorithms (2nd ed.), Cambridge University Press, pp. 46–48, 2011.
29. Paolo Tasca, Thayabaran Thanabalasingham, Claudio J. Tessone. Taxonomy of Blockchain Technologies. Principles of Identification and Classification. Forthcoming Ledger Journal. 2018.
30. Back, Adam. Hashcash-a denial of service counter-measure. 2002.
31. David Schwartz, Noah Youngs, Arthur Britto. The Ripple Protocol Consensus Algorithm. 2014.

32. Juri Mattila. The Blockchain Phenomenon. In: The Blockchain Phenomenon(Berkeley Roundtable of the International Economy, 2016, edn.). 2016.
33. Peter Todd. Soft Forks Are Safer Than Hard Forks. 2016. [ONLINE] Available at https://petertodd.org/2016/soft-forks-are-safer-than-hard-forks [Accessed 10 January 2018]
34. Haerder, T.; Reuter, A.. Principles of transaction-oriented database recovery. ACM Computing Surveys. 15 (4): 287. doi:10.1145/289.291. 1983
35. Brewer, Eric A. Towards robust distributed systems. PODC. Vol. 7. 2000.
36. Bitcoin Wiki. Colored Coins. https://en.bitcoin.it/wiki/Colored_Coins, 2018. [Accessed 03 January 2018]
37. Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman.: Concurrency Control and Recovery in Database Systems, Chapter 7, Addison Wesley Publishing Company. 1987.
38. Rob Price, Digital Currency Ethereum Is Cratering because of a $50 Million Hack. Business Insider, 16 Jun. 2016; 13 [ONLINE] Available at http://uk.businessinsider.com/dao-hacked-ethereum-crashing-in-value-tens-of-millions-allegedly-stolen-2016-6?IR=T [Accessed 01 December 2017].
39. Croman, Kyle, et al. On scaling decentralised blockchains. International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2016.
40. Paolo Tasca, and Sebastian Widmann. The challenges faced by blockchain technologies–Part 1. Journal of Digital Banking 2.2 (2017): 132-147.
41. Paolo Tasca, and Sebastian Widmann. The challenges faced by blockchain technologies–Part 2. Forthcoming *Journal of Digital Banking*. 2018.