Untangling the overlap between Blockchain and DLTs

Badr BELLAJ[1][2], Aafaf OUADDAH[1*],Emmanuel BERTIN[3],Noel Crespi[2] and Abdellatif MEZRIOUI[1]

[1] National Institute of Post and Telecommunication (INPT),Rabat,Morocco
aafafouaddah@gmail.com
[2] Institut polytechnique de Paris, Paris,France
[3] Orange Lab, Caen, France

Abstract. The proven ability of Bitcoin and other cryptocurrencies to operate autonomously on the trustless Internet has sparked a big interest in the underlying technology – Blockchain. However, the portage of Blockchain technology outside its initial use case led to the inception of new types of Blockchains adapted to different specifications and with different designs. This unplanned evolution resulted in multiple definitions of what a Blockchain is. The technology has diverged from its baseline (Bitcoin) to the point where some systems marketed as "blockchain" share only a few design concepts with the original Blockchain design. This conceptual divergence alongside the lack of comprehensive models and standards made it difficult for both system designers and decision-makers to clearly understand what is a blockchain or to choose a suitable blockchain solution. To tackle this issue, we propose in this paper "DCEA" a holistic reference model for conceptualizing and analysing blockchains and distributed ledger technologies (DLT) using a layer-wise framework that envisions all these systems as constructed of four layers: the data, consensus, execution and application layers.

Keywords: Blockchain, DLT, Reference model, blockchain-like, review

1 Introduction

The emergence of many projects heavily inspired by Bitcoin's blockchain drove the industry to adopt a broader term; "Distributed ledger technology", or DLT when referring to this category. Nevertheless, there is no rigorously defined set of terminologies or commonly acceptable reference model delineating the border of DLTs subcategories. As a result, terms like "blockchain", "DLT" or even "distributed database" were misunderstood, misused, and misinterpreted. As a result, many projects or enterprises use the word "blockchain" extensively, simply as a marketing word to describe their Blockchain like products without abiding by a clear standard. Although there are multiple proposals for standardizing blockchain (ISO [6]-[9], IEEE [3], ITU [10]), there is no recognized standard for defining blockchain or DLT. In the absence of a referential definition, we observe the growing use of imprecise and inconsistent language and terminology across different projects— where the same term may be used to refer to different things— that leads usually to confusion. To help untangle the underlying concepts and delineate the different categories of DLTs, we define in this paper a reference model capturing a longitudinal and representative view of DLT systems. The proposed model introduces a systematic and holistic approach to conceptualizing and analysing DLTs in general as a functioning system constructed of four key layers: Data, consensus, execution and application layers.

Hence, the rest of the paper is structured as follows. Section 2 defines a new layer-wised framework serving to normalize and deliberates on the classification and taxonomy of DLTs Sections 3, 4, 5 and 6 present, respectively the data, the consensus, the execution and the application layers, where in each section we outline the main components and properties of the studied layer as well as it related state of the art. In section7 we discuss the briefly the difference between Blockchain and Blockchain-like systems. Finally, we close with a conclusion in section 8.

2 DCEA framework

We propose DCEA, a framework that defines a layered heterogeneous stack for DLT systems. From a design perspective, our conceptual framework (DCEA) segregates DLT technologies into four essential and distinct layers: data, consensus, execution and application layers — each one playing a well-defined role in the DLT architecture. The framework consists of the DLTs components and their main properties (Table 1, with logically related functions grouped together. This layering approach is aligned with the DLT's modular architecture. It will help to provide a better understanding of DLTs and serves as a baseline to build a comparative analogy between different DLT variants.

In the following, we introduce the four layers that form the DLT stack.

– Data Layer: Represents the data (transactions and states) flowing through the distributed network and stored in the ledger. Data in this layer is represented by entries recorded in the ledger, under consensus and shared amongst the network participants. These records may represent elements defined by the underlying protocols (such as cryptocurrency, or smart contracts), or data received from external environments (such as IoT data). Generally, the data layer covers data stored on the blockchain itself (on-chain storage) as well as data stored in an auxiliary source using a distributed database (off-chain storage).

Table 1. Layers and components of DCEA framework

Application Layer	Integrability			DLT orientation and purpose			Wallet and identity management		
Execution Layer	Execution environment		Turing-completeness		Determinism		Openness	Interoperability	
Consensus Layer	Safety	Liveness	Finality	Network model	Failure model	Adversary model	Governance model	Transaction ordering	Conflict resolution
Data Layer	Data structure		Data shareability		Data immutability		States storage		

– Consensus layer: Defines the global software-defined ruleset to ensure agreement among all network participants on a unified ledger. Consequently, this layer designates the formal rules that govern the system.

- Execution layer: Represents the components responsible for enforcing and executing distributed programs (e.g. smart contracts). Basically, these programs or contracts codify a given logic (e.g. a business logic) as a set of instructions for manipulating the states recorded in the ledger.
- Application layer: Represents an abstraction layer that specifies a variety of protocols and APIs provided by the DLT system to enable the building of distributed applications commonly called DApps. This layer also represents a communication link between the external actors or applications and the code hosted on the DLT ledger.

Based on the above layering, we propose a four-layered taxonomy (Table 1), to categorize DLT systems. The purpose of the taxonomy is to:

- Classify the typical DLT systems proposed in the academia and in the industry; and to
- the relative strengths and weaknesses of existing systems and identify deficiencies in the current set of DLTs.

At each layer, DLTs adopt different settings for DCEA properties defined in Table (1). Based on their combinations, at the four layers, we can define different DLT classes. For instance, at the data layer we differentiate between DAG-based and chain-based DLTs based on the nature of the underlying data structure; at the consensus layer we differentiate between permissioned and permissionless DLTs based on the identity model of the consensus mechanism; at the execution layer we differentiate between Smart-contract based DLTs and script-based DLTs; and at the application layer, we can differentiate between DApps-oriented DLTs and Cryptocurrency-oriented.

3 DATA LAYER

In this section, we lay out the key components, and their characteristics, that construct the data layer as introduced in Fig 1.

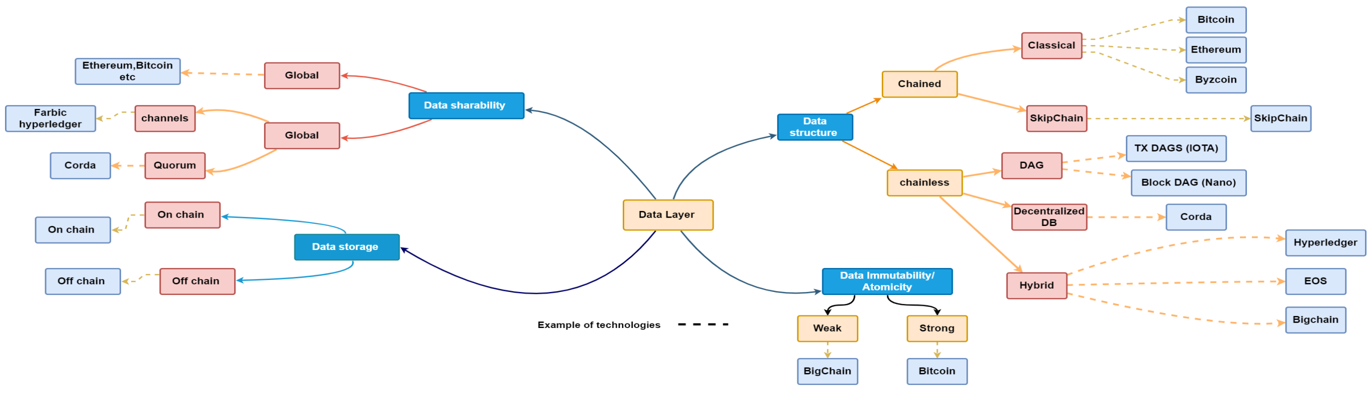

Fig. 1. The main components of the data layer with examples

3.1 Components and properties

DLT's ledger represents a distributed data store where data is duplicated among multiple nodes, by means of data synchronization. In these data stores, the data organization ,in its macroscopic structure, varies from one technology to another. Generally, we distinguish between two main models of data structures in the DLT space; the linear chain of blocks and the chain-less models.

Chained model

Chain of blocks: Data in the chain of blocks is organized in elementary units called blocks. Each block is a collection of transactions validated by the network. These units are organized chronologically as a chain of inter-hinged blocks, which are tied by tamper-evident hash pointers. Each new block can only be valid if it is built upon an unchangeable body of previous blocks. Blocks are composed of a header and a record of transactions. The block's header contains meaningful metadata such as a cryptographic reference to the previous block and the current time. This linear linkability ensures data integrity through cryptographic connections between blocks and enables each participant in the network to verify and validate data. Data in a chain of blocks is carried over and stored in the ledger using transactions, therefore we consider a transaction as the most elementary data type. At the block level, the transactions are ordered and hashed into a Merkle tree, with the hash root placed in the corresponding block's header. This structure guarantees a cryptographically sealed and tamper-proof data vault resistant to any type of data corruption.
Skipchain: The data structure of a skipchain is inspired by skip lists . Skipchain adapts the skip list idea to the chain of blocks by adding links between blocks both forwards and backwards in time. In skipchain, a block includes not just a single hash link to the previous block, but also an additional hash link to a point farther back in time. Thus, skipchain can build subsequent layers of linked blocks on top of an original linked list of blocks. Skipchain is very useful when concurrent access to data is required.

Chainless model

In order to overcome some limitations imposed by the adoption of the chained block structure, certain DLTs have opted for a chain-less model. Instead, they use new data structures for better scalability or security.
DAG: In contrast to using a chain of blocks, some DLTs are using a nonlinear structure such as the Direct Acyclic Graph (DAG) to offer better performance. A DAG is a graph that grows in one direction without cycles connecting the other edges (i.e., there is no path from a vertex back to itself). As with a chain of blocks, a DAG is used as a sorted collection of hierarchically connected transactions where each transaction and sometimes a block of transactions is represented by a node (which refers to a vertex in the graph) and linked to at least another node. The DAG is extended chronologically by appending new transactions to the previous nodes. The ledger is thus an ever-growing tree, starting initially from a root node. The acyclic nature of the DAG and its unidirectional evolution enables participants to confirm transactions automatically based on previous transactions. Based on the representation of its nodes, we identify two types of DAGs:

– Transaction-based DAGs: DAG nodes that represent individual transactions; and
– Block-based DAGs: DAG nodes that represent a block of transactions.

Decentralized database model: Some DLTs adopt radical changes in their architecture over the conventional blockchains, to the point they resemble a classical distributed database. We consider these solutions as decentralized databases, as they manage data similar to how conventional databases handle data but they present a different technology. In fact, unlike in a conventional distributed database, where nodes cooperate to maintain a consistent view of data across all systems, a decentralized database is designed to allow multiple parties that may not trust each other to collaborate with their peers to maintain shared records.

Hybrid data model: Some DLT projects combine both chains of block models along a block-less model to manage transactions and states in the network. The hybridization is designed to exploit the advantages of each model to enable better scalability and rapid transaction validation. In this model, the states are generally stored in external dedicated key-value databases and the blocks contain only the transactions affecting the ledger's states. Using key-value databases makes it easy to directly access the updated value of a state rather than having to calculate it by traversing trees of transactions.

State management A key distinguishing factor among various DLTs, is how states are managed within the system. Although DLTs serve as distributed ledgers for shared data, in the case of many DLTs, data is stored outside the transactional distributed ledger (off-chain/off-ledger) using auxiliary databases. Conventional blockchains, however, tend to always store data on the shared ledger (on-chain/on-ledger). When we analyze how general states (e.g. user's balance) are managed in existing DLTs, two models emerge UTXO model, Account model. The first is a special transactions set, linking new transactions to old transactions, wherein a newly produced transaction (new UTXO) points to a single or multiple ulterior transactions (inputs), whereas, the second is a model where the ledger keeps track of up-to-date global states related to each account.

Data shareability All nodes in a DLT network exchange transactions carrying shared data in order to reach consensus, but due to privacy reasons different visions of data shareability have been adopted. Some systems favor complete shareability of all data — which we consider as global shareability—, whereas others restrict the perimeter of shareability including some nodes and excluding others — which we consider as restricted shareability.

Data immutability / Atomicity There is a common belief that records stored on a DLT (especially a blockchain) are immutable and unalterable. However, that is not necessarily the case, as different DLT systems provide different degrees of immutability depending on the system design. This means that, under some circumstances, nodes can hold inconsistent states, or that a confirmed transaction may be reversed. For data immutability, we differentiate between:

– Strong immutability. When the state variables or blockchain entries cannot be mutated or tampered after their creation; and
– Weak immutability. When the state variables or blockchain entries could be mutated or tampered with after their creation.

It is worth noting that for some strong immutable systems, their states can be updated without breaking immutability. This is achieved by using tree structures to store persistently both new and old values for a given entry.

Data privacy Data privacy is securing data from public view. In a shared context like in DLTs, data can be private or not private. This is possible with cryptographic techniques such as Zero-knowledge proofs which enable verifying private data without revealing it in its clear form.

3.2 Data layer: state of the art

This subsection is meant to present an overview of DLTs projects adopting the different data structures previously outlined by our framework as well as an evaluation of their properties.

Chained DLTs Most DLTs follow the linear data chain structure initially defined by Bitcoin. In this broad category, multiple projects define different inner block structures.

Bitcoin In Bitcoin and its clones, transactions are assembled in the block's body and then linked in a Merkle tree. The root of this tree, or the Merkle root, is a hash representing an indirect hash of all the hashed pairs of transactions in the tree and is included in the block header, thereby ensuring transaction verification. In addition to the Merkle root, the block header also contains other important information, including: the timestamp, and the previous block's hash. Moreover, Bitcoin adopts the UTXO model to track the system states (Wallet balances). The UTXO set is stored off-chain in an auxiliary database.

Ethereum The block structure is more complex in Ethereum than in Bitcoin, and the system's state tracking is different than in Bitcoin. In fact, the block's header comprises more metadata and its body englobe multiple types of data, namely: transactions, receipts and system states. Each of these data types is organized into a Merkle tree or a Patricia tree (Radix tree) in the case of the state tree. The state tree is an important component in the Ethereum ledger, as it is used to implement the account model, whereby each account is linked to its related states (account balances, smart contract states, etc.). Any node can parse the tree and get the updated state without any overhead calculation. The state tree grows each time a change occurs in a state. It grows by adding new nodes (stored in the new block) — containing new states— which points to the nodes (stored in the previous block) containing the old value for the same state. To enforce immutability Ethereum keeps its root hash in the block header.

Skipchain Chainiac Nikitin and al. [17] introduced Chainiac to solve offline transaction verification problems (enable nodes to check if a transaction has been committed to a blockchain without having a full copy of the ledger). The Chainiac solution was to add traversability forward in time using a skipchain, where back-pointers in Chainiac are crypto-graphic hashes, whereas forward-pointers are collective signatures. With long-distance forward links and via collective signatures, a client or node can efficiently verify a transaction anywhere in time.

Chainless DLT

DAG based chains The idea of using DAGs as underlying data structure has encountered great interest from DLT designers of multiple projects, including Byteball, DagCoin IOTA NanoPhantom and Hedera . Some studies have tried to introduce DAG in conventional blockchain DLTs, for instance the GHOST protocol [12] proposes a modification of the Bitcoin protocol by making the main ledger a tree instead of a blockchain. Such a modification reduces confirmation times and improves the overall security of the network.

Decentralized Databases: Corda R3 In the corda network, each node maintains a local database called a "vault" that stores time-stamped data. Each vault has many different versions (current and historic) of data in the form of state objects. A vault does not store transactions, instead it stores the output state relevant to a party (state's participants). The transactions are stored in the "NODE1_TRANSACTIONS" table in the node's database . Alongside, Corda adopts a UTXO model to store state data, which means a transaction consumes current states and produces or not new states.

Hybrid DLTs: Hyperledger Fabric Hyperledger Fabric combines between the usage of a chain of blocks to store only the validated transactions, and the usage of a key-value classical database to store the system's states (transaction outcomes). In the Fabric chain, the block structure resembles the structure of a block in a conventional chain but with an additional part: block metadata. This additional section contains a timestamp, as well as the certificate, public key and signature of the block writer. The block header is straightforward and the transactions are ordered in the block body without Merkilization.

BigchainDB The BigchainDB [15] was introduced as a blockchain database. It aims to combine the key characteristics of "traditional" NoSQL databases (MongoDb) and the key benefits of traditional blockchains. BigchainDB server nodes utilize two distributed databases (transaction set or "backlog") holding incoming transactions and a chain of blocks storing validated transactions (Creation or Transfers). Each transaction represents an immutable asset (represented as JSON documents in MongoDB).

Data shareability Most DLTs operating as global cryptocurrency platforms adopt by design a global shareability of the transactions. In fact, networks such as Bitcoin, Ethereum and many others, operate in relay mode where nodes are relaying transactions to each other, thereby propagating it to the entire network without restrictions. In other DLTs, such as Hashgraph, senders deliver their transactions to a set of selected nodes that are responsible for including them into their DAG and sharing them with others by Gossiping. On the other hand, the DLTs constructed for business purposes, such as Corda or Hyperledger Fabric, impose restricted shareability of the transactions as privacy is an important requirement in such contexts. In Corda, for instance, each node maintains a separate database of data that is only relevant to it. As a result, each peer sees only a subset of the ledger, and no peer is aware of the ledger in its entirety. In Fabric a subset of the ledger restricts data shareability by using the concept of channels [5]. A channel is a private sub network between two or more specific network members. Each transaction on the network is executed on a channel, where only authenticated and authorized parties are able to transact on that channel. Therefore, the network ends up with a different ledger for each channel. Similarly, Quorum, an Ethereum-based distributed ledger protocol with transaction/contract privacy, enables sending private transactions between multiple parties in the network by use of constellations.

4 Consensus Layer

DLTs have renewed the interest in the design of new distributed consensus protocols. In fact, a myriad of consensus algorithms, for DLT, have been proposed in the literature presenting different properties and functionalities. In this section, we present the properties and features we consider as part of the DCEA framework for studying and differentiating between the protocols.

4.1 Components and properties

Basic Properties The concepts of safety and liveness properties were introduced initially by Lamport in 1977 and have been well adopted in the distributed computing community. All consensus algorithms provide these properties under different assumptions of synchrony, adversary model, etc.

Safety Safety represents in the context of DLT networks, the guarantee that the correct nodes will not validate conflicting outputs (or make conflicting decisions) at the same time (e.g. chain forks).

Liveness A consensus protocol guarantees liveness if requests (transactions) from correct clients are eventually processed.

Finality In the DLT settings, we define the finality property as the affirmation and the guarantee for a transaction to be considered by the system as final and irreversible. The Finality as property can be divided into two types:

- Probabilistic finality, where the probability that a validated transaction will not be reverted, increases with time after the transaction is recorded onto the ledger.
- Absolute finality, where a transaction is considered finalized once it is validated by the honest majority.

Network models In both traditional distributed systems literature and DLT consensus protocols, we consider the message-passing model in which nodes exchange messages over the network, under differing assumptions of network synchrony. We adopt in this survey the following taxonomy defined by [7].

- Synchronous, where we assume the existence of a known upper bound on message delay. That means messages are always delivered within sometime after being sent.
- Partially-synchronous, where we assume there is some known Global Stabilization Time (GST), after which the messages sent are received by their recipients within some fixed time-bound. Before the GST, the messages may be delayed arbitrarily.
- Asynchronous, where messages sent by parties are eventually delivered. They may be arbitrarily delayed and no bound is assumed on the delay of messages to be delivered.

Failure Models Different failure models have been considered in the literature; we list hereafter two major types.

- Fail stop failure (Also known as benign or crash faults): Where nodes go offline because of a hardware or software crash.
- Byzantine faults: This category of faults was introduced and characterized by Leslie Lamport in the Byzantine Generals Problem to represent nodes behaving arbitrarily due to software bugs or a malicious compromise. A Byzantine node may take arbitrary actions, provide ambivalent responses or intentionally mislead other nodes by sending sequences of messages that can defeat properties of the consensus protocol.

We consider, therefore, a protocol as fault tolerant, if it can gracefully continue operating without interruption in the presence of failing nodes.

Adversary models Under the assumption of a message-passing model, the adversary is able to learn the message exchange and to corrupt different parts of the network. We distinguish between the following three adversary models:

- The Threshold Adversary Model : This model is the most common adversary assumption used in the traditional distributed computing literature, which assumes that the Byzantine adversary can corrupt up to any f nodes among a fixed set of n nodes. Under this model, the network usually has a closed membership requiring a permission to join. The consensus protocol should be able to operate correctly and reach consensus in the presence of Byzantine nodes as long as their numbers do not exceed a given threshold.
- Computational Threshold Adversary: A new model introduced by Bitcoin, where the control of the adversary over the network is bounded by the computational power –requiring concrete computational material— instead of the number of nodes he can control. In this model, typically the membership is open and multiples parties and the bounding computation is a brute force calculation.
- Stake Threshold Adversary [1]: In this model, the adversary control is bound by his proportion of a finite financial resource. In networks managing cryptocurrencies,

the underlying protocol can ensure consensus based on cryptocurrency deposits, thus the adversary is bound by the share of cryptocurrency he owns. In addition, in these protocols' punishment rules (e.g. stake slashing) could be put in place to deter bad behaviour.

Adversary Modes Consensus protocols assume the existence of different types of adversaries based on their ability and the time they need to corrupt a node.

– Static adversary: A Byzantine user who is able to corrupt a certain number of network nodes ahead of time and exercise complete control over them. However, he is not able to change which nodes they have corrupted or to corrupt new nodes over time.
– Adaptive adversary: A Byzantine user who has the ability to control nodes and dynamically change, depending on the circumstances, the nodes under his control to gain more power.
– Mildly adaptive adversary: A Byzantine user who can only corrupt nodes based on past messages, or its anticipations, and cannot alter messages already sent. Moreover, the adversary may mildly corrupt groups, but this corruption takes longer than the activity period of the group.
– Strongly adaptive adversary: A Byzantine user can learn of all messages sent by honest parties, and based on their content, he can decide whether or not to corrupt a party by altering its message or delaying message delivery.

Identity Model Protocols manage nodes membership differently, but in general two opposite sides are adopted:

– Permissionless, where the membership is open and any node can join the network and validate new entries.
– Permissioned, where the membership is closed and only a restricted set of approved members is able to validate new entries.

In the DLT settings, the identity model is commonly bound to the network openness nature —being private, public or consortium. .

Governance Model The governance model refers to the process of decision-making adopted by a DLT network to decide on the protocol rules and their upgrade. Hence, the governance of the system boils down to a social concept, we find it appropriate to identify some of the possible governance models from a social perspective:

– Anarchic, where protocols upgrade proposals are approved by every participant in the network. Each participant chooses to accept or reject a given proposal, thus leading to potential splits in the network.
– Democratic, where participants vote on new rules and protocol upgrades proposals and at the end, all participants have to follow the decision of the majority, even for those participants who voted against them.
– Oligarchic, where new rules and protocol upgrades are proposed and approved by a group of participants.

As most DLTs move governance and related issues “on-chain” or “off-chain” we consider also the differentiation between; Built-in (or on-chain governance), where the decision-making process in the network is defined as part of the underlying consensus protocol; External governance (or off-chain governance), where the decision-making process is based on procedures independently performed without involving the DLT mechanisms.

Transactions ordering Whether for a linear or a non-linear DLT (e.g. DAGs), the stored transaction should be ordered chronologically to avoid frauds and inconsistencies. Different approaches have been introduced by the consensus protocols to provide reliable and fair transaction ordering. Usually, in DLTs the ordering is an integrating part of the consensus mechanism but in some cases, it can be decoupled from the execution and validation of transactions. Ordering is an important property with direct impacts on the security and the usage of a DLT, thus the need to evaluate this feature separately.

Conflict resolution model In some DLT networks, conflicting temporary versions of the ledger (known as forks) can coexist for different reasons (e.g. network latency, parallel validation of blocks, etc.). To converge toward a canonical ledger or chain, networks and consensus mechanisms adopt different rules. The most notable rule is defined by Bitcoin protocol as the “longest chain rule”, whereby in the presence of conflicting orders, the network converges to one order following the longest chain — the chain with the largest accumulated PoW in case of PoW-based systems— and discards the rest. The longest chain rule is adopted by different protocols and each may adopt a different cumulative parameter (witnesses votes, endorsement, etc.).

4.2 Consensus layer: state of the art

In this subsection, we present multiple consensus mechanisms and their properties. Although, is out of scope of this paper to present a detailed taxonomy of the existing protocols (fig. 2), we consider to group all the reviewed protocols in six categories. This protocol categorization serves us as a basis to categorize the DLTs.

BFT consensus family (PBFT-like) This family refers to the classical consensus mechanisms introduced in the traditional distributed computing literature and their recent variants. The BFT family is easily recognized due to their property: all-to-all voting rounds , the identity of the nodes in the network is known, the number

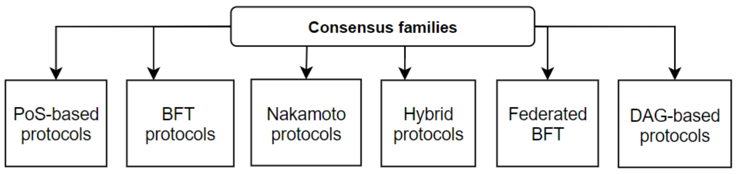

Fig. 2. Taxonomy of consensus protocols

of participants is limited. Due to the big number of the protocols belonging to this family, we limit our review, in this paper, on the most used algorithms in DLT context namely; PBFT, RAFT, IBFT, DBFT, POA (AURA, Clique), HoneyBadgerBFT and Hotstuff.

Nakamoto consensus family We consider that Nakamoto's consensus family represents protocols using a chain of block data structure and adopting the longest chain fork choice rule (or a variant like GHOST [21]), to ensure safety, along economic incentives. These protocols were introduced primarily to enable secure currency transfer over the internet. Conversely to PBFT, they are conceptually simple and tolerate important corruptions up to *n/2*. Besides, they are known for being permission-less (open enrollment) — they do not require node authentication and allow nodes to arbitrarily join or leave the network. We review hereafter some of the most discussed protocols in this category namely: PoW, memory bound PoW and BitcoinNG.

Proof of stake and its variants Proof-of-Stake (PoS) was first proposed as an alternative to the costly PoW for use in the PPCoin [13]. Instead of a hash-calculation competition between validators, participants who desire to join validators board and forge the new block have to lock a certain amount of coins into the network as a financial stake. Thus, the chances for a node to be selected as the next validator depends on the size of the stake. Different implementations of PoS exist. We present here some of the typical representatives including Ethereum PoS, DPoS (EOS), Ouroboros and its variants, and Snow white.

Hybrid protocols This family represents protocols which attempt to benefit from the advantages of the known protocols PoW, PoS and other established protocols to provide better performance.

DAG-based Protocols IOTA is a hybrid consensus protocol, marrying between PoW at the entry level and a custom transaction validation algorithm. IOTA relies on PoW to protect the network against spamming as the transactions are fee-less. paragraphAvalanche Avalanche is a recent leaderless Byzantine fault tolerance protocol built on a metastable mechanism via network subsampling. Avalanche protocol is based on a metastable mechanism, whereby a node repeatedly takes a uniform random sample from the network, sends queries repeatedly in multiple rounds and collects responses.

Federated BFT Ripple was the first implementation of a federated Byzantine agreement system (FBAS, for short), which was extended later by Stellar protocol. FBA revisits BFT settings by providing an open membership service based on a trust model. In fact, FBA protocols depart from the concept that each node interacts only with a limited group of its trusted peers — the unique node list (UNL) in Ripple and the quorum slice in Stellar. Thus, unlike traditional BFT protocols, the federated Byzantine agreement (FBA) does not require a global and unanimous agreement among the network participants.

5 Execution Layer

In this section,as illustrated in Fig 3, we identify the fundamental components of the execution layer and their properties. Then we present the execution component widely adopted in the state-of-the-art.

5.1 Components and properties

In a DLT system, business logic, agreed to by counterparties, can be codified using a set of instructions and embedded into the ledger in a specific format. The ruleset execution is enforced by the distributed consensus mechanism. Generally, we distinguish between two main models for rules codification: Smart contracts and built-in scripts (scripting model).

Execution environment

Smart contract model In this model, clauses between counterparties are codified as a stateful self-executed program. Typically, this program (known as smart contract) is implemented either in a dedicated language or using an existing programming language such as Java or C++. The smart contract execution is handled by a dedicated environment such as a virtual machine or a compiler, which proceeds the instructions defined in the triggering transaction, returns an output and often results in updating states. Commonly, the smart contracts live and execute on the DLT as an independent entity with reserved states. Although they are qualified as 'smart', they are not autonomous programs, as they need external triggering transactions, nor contracts in a legal sense.

Scripting model Unlike the smart contract model, the scripting model enables codifying the desired logic using only a usage-oriented and predefined set of rules defined by the protocol, which limits the possible scenarios to implement. The idea behind this limitation is to avoid security problems and reduce the complexity of the system. Typically, scripting model is implemented in the DLTs that focuses on securing the manipulation of built-in assets rather than providing a platform for running universal programs.

Turing completeness Generally speaking, a given environment or programming language is said to be Turing-complete if it is computationally equivalent to a Turing machine . That is, a Turing-complete smart contract language or environment is capable of performing any possible calculation using a finite amount of resources. Some DLTs are capable of supporting a Turing-complete execution environment, which provides its users with the flexibility to define complex smart contracts, whereas other DLTs provides Non-Turing complete execution environments because they suffer from some inherent limitations.

Determinism Determinism is an essential characteristic of the execution environment in DLT systems. Since the distributed program (e.g. smart contract) is executed across multiple nodes, the deterministic behaviour is needed to yield coherent and identical outputs to obviate discrepancies in the network. In order to ensure determinism, DLTs have to handle non-deterministic operations (e.g. Floating-point arithmetic, or random number generation, etc.) either by disabling these features or by enabling them in a controlled environment.

Runtime openness In most DLTs, the execution environment or runtime is by design an isolated component without connections with external networks (e.g. Internet). However, in many case scenarios, the need for accessing information from outside the DLT manifested as a necessity. Thus, to allow such a feature, different design choices were introduced which can be classified into three approaches:

- Isolated: where interactions between the smart contract execution environment and the external environments are not allowed
- Oracle-based: where interactions with external environments are managed by members of the network who are called oracles. An oracle refers to a third-party or a decentralized data feed service that provides external data to the network.
- Open: The execution layer is able to connect to the external environments.

Interoperability DLT operating networks are currently by-design siloed and isolated from each other. The interoperability, which we consider as the ability to exchange data, assets or transactions between different DLTs, is a complex operation that requires passing transactions between them, in a trustless manner without the intervention of third parties. Interoperability is a highly desired property thus multiple solutions were developed to enable interoperability between different existing DLTs. These solutions can be categorized into the following groups:

- Sidechain : is a blockchain running in parallel with another chain (known as main chain) that allows transferring data (cryptocurrency) from the main chain to itself.
- Multichain : is a network of interconnecting chains, upon which other chains can be built. In a multichain one major ledger rules all the sub-ledgers.
- Interoperability protocols: represent protocols and means (e.g. smart contracts) added to the original DLT to enable interoperability with other DLTs.
- Interoperable DLT: represents a DLT designed with the goal to enable interoperability between other DLTS.

5.2 Execution layer: state of the art

In this section, we provide an overview of the most widely-used execution environments implemented in the industry and the literature with a discussion of their properties.

Execution environments

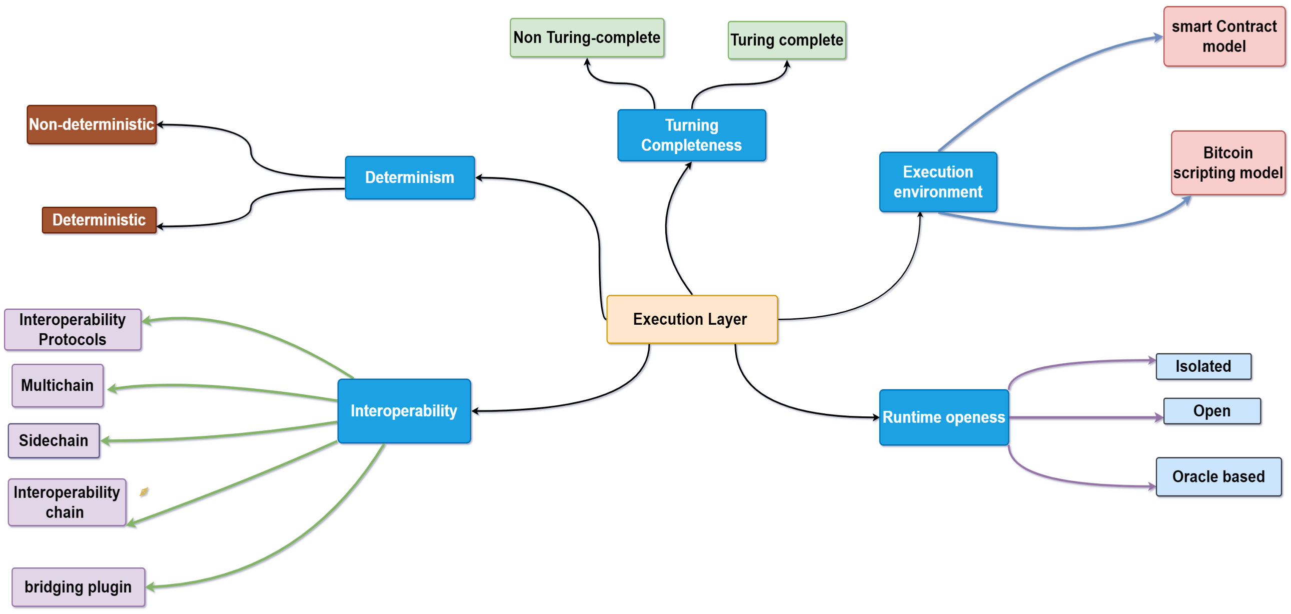

Fig. 3. The main components of execution layer

Ethereum Virtual machine In Ethereum, smart contracts represent a computer program written in a high-level language (e.g. Solidity, LLL, Viper, Bamboo, etc.) and compiled into a low-level machine bytecode using an Ethereum compiler. This bytecode is stored in a dedicated account —therefore has an address— in the blockchain. Then, it is loaded and run reliably in a stack-based virtual machine called Ethereum Virtual Machine (EVM for short), by each validating node when it is invoked. To enable the execution of the bytecode and state update, the EVM operates as a stack-based virtual machine. It uses a 256-bit register stack from which the most recent 16 items can be accessed or manipulated at once. The stack has a maximum size of 1024 possible entries of 256-bits words. The EVM has a volatile memory operating as a word-addressed byte array, where each byte is assigned its own memory address. The EVM has also a persistent storage space which is a word-addressable word array. The EVM storage is a key-value mapping of 2^{256} slots of 32 bytes each. Unlike the memory which is volatile, storage is non-volatile and it is maintained as part of the system state. The EVM is a sandboxed runtime and a completely isolated environment. That is, every smart contract running inside the EVM has no access to the network, file system, or other processes running on the computer hosting the EVM. The EVM is a security-oriented virtual machine, designed to permit the execution of unsafe code. Thus, to prevent Denial-of-Service (DoS) attack, EVM adopts the gas system, whereby every computation of a program must be paid for upfront in a dedicated unit called gas as defined by the protocol. If the provided amount of gas does not cover the cost of execution, the transaction fails. Assuming given enough memory and gas, the EVM can be considered a Turing-complete machine as it enables to perform all sorts of calculations.

Bitcoin scripting Bitcoin uses a simple stack-based machine to execute Bitcoin scripts. A Bitcoin script is written using a basic Forth-like language. It consists of a sequence of instructions (opcodes), loaded into a stack and executed sequentially. The script is run from left-to-right using a push-pop stack. A script is valid if the top stack item is true (non-zero) at the end of its execution. Bitcoin scripting is intentionally not Turing-complete, with no loops. Moreover, the execution time is bounded by the length of the script (Maximum 10 kilobytes long after the instruction pointer. This limitation prevents denial of service attacks on nodes validating the blocks. Bitcoin scripting is considered as a limited and complex language for writing smart contracts, to overcome this limitation, multiple projects have been introduced, such as Ivy [11] , Simplicity , BitMLwhich are high-language with richer features that compile into Bitcoin scripts. Also, [4] introduced BALZaC - a high-level language based on the formal model, and Miniscript was proposed recently as a language for writing (a subset of) Bitcoin Scripts in a structured way, enabling analysis, composition, generic signing and other features. In addition, Rootstock (RSK)[16] was proposed as a smart-contract platform that integrates an Ethereum-compatible virtual machine to Bitcoin.

Interoperability The inability for siloed DLTs to communicate with one another has been a major hindrance to the development of the blockchain space, therefore different proposals have aimed to solve this problem. In this subsection we present the most important approaches implemented at the execution layer to solve this problem.

Sidechains Multiple sidechains have been proposed in the DLT ecosystem. Rootstock is a sidechain of Bitcoin, equipped with RVM a built-in compatible Ethereum virtual machine. Rootstock chain is connected to the Bitcoin (BTC) blockchain via a two-way peg enabling transfers from BTC to SBTC (Rootstock's built-in currency) and vice versa using Bitcoin scripts, whereby users lock up their BTC in a special address and get an equivalent amount of RBTC on the sidechain. Similarly, Counterparty is another sidechain of Bitcoin where coins to be transferred are burned or locked by sending them to an unspendable address and generating the equivalent in the Counterparty chain. Drivechain is another proposal for transferring BTC between Bitcoin blockchain and sidechains. Unlike most DLTs where the sidechain is a separate project, Cardano has introduced Cardano KMZ sidechain as part of its ecosystem. Cardano KMZ is a protocol which serves for moving assets from its two-layer CSL to the CCL (Cardano Computation Layer), or other blockchains that support the Cardano KMZ protocol. Another sidechain-based project is Plasma [19]. It aims for creating hierarchical trees of sidechains (or child blockchains) using a combination of smart contracts running on the root chain (Ethereum). The idea is to build connected and interoperable chains operated by individuals or a group of validators rather than by the entire underlying network. Thus, Plasma helps scaling Ethereum by moving transactions toward the sidechains. Currently, Plasma is actively developed and used by projects such as OmiseGo , which aims to build a peer-to-peer decentralized exchange, and Loom , which provides the tools needed to build high-performance DApps while being operating over the Ethereum network.

Determinism To deal with the non-determinism issue, three general approaches are adopted. The first approach is to guarantee determinism by design. For instance, in Ethereum the EVM does not support, by-design, any non-deterministic operations (e.g. floating point, randomness, etc.). Nevertheless, due to the importance of randomness, the RANDAO [20] project has been proposed as an RNG (Random number generator) of Ethereum based on an economically secure coin toss protocol. The idea behind is to build DAO (decentralized autonomous organisation) for registering random data on the blockchain. The second approach, adopted by other projects such as Multichain , Corda, or Stratis which use existing runtime environments, ensure determinism by adapting these environments to force determinism processing. For instance, MultiChain uses Google's V8 engine with sources of non-determinism disabled. Similarly, corda uses a custom-built JVM sandbox . Stratis limits the capabilities of C#, or all of .NET's core libraries that can be used . The third approach (Determinism by endorsement) introduced by Hyperledger Fabric ensures determinism differently. In order to guarantee determinism, the endorsement policy can specify the endorsing nodes to simulate the transactions and execute the Chaincode. In the case, where endorsing peers diverge with different outputs, the endorsement policy fails and the results will not be committed into the ledger.

5.3 Environment openness

Most DLTs rely on oracles to read data from external sources. Simply put, an oracle is a smart contract maintained by an operator that is able to interact with the outside world. Several data feeds are deployed today for smart contract systems such as Ethereum. Examples include , Town Crier [24] and oracle Oraclize.it . The latter relies on the reputation of the service provider and the former is based on the concept of enclave hardware root of trust [8]. Other oracles such as Gnosis and Augur [18], leverages prediction markets MakerDao , which is a decentralized lending facility built on the Ethereum blockchain, utilizes a multi-tiered approach to feed reliable price data for its assets without sacrificing the decentralization. For instance, Medianizer [14] is used as a MakerDao oracle, which serves to provide accurate prices for Ethereum, collects data from 14 independent price feeds. Similar to the MakerDAO system, ChainLink [22] aggregates data feeds from many sources. Conversely to most DLTs, Fabric's Chaincode is able to interact with external sources such as an online HTTP or REST API . In case, where every endorser gets a different answer from the called API, the endorsement policy will fail and, therefore no transaction will take place. Other DLTs, like Aeternity [2] incorporate an oracle in the blockchain consensus mechanism removing the need for a third-party.

6 Application layer

In this section, we briefly introduce the components and properties we defines for the application layer and review the related solution as illustarted in Fig 4.

6.1 components and properties

Integrability As a new technology, which is often perceived as hard to adopt, DLT systems try to offer a better user-experience by providing necessary tools (APIs, frameworks, protocols.) to enable better integrability with existing technologies and

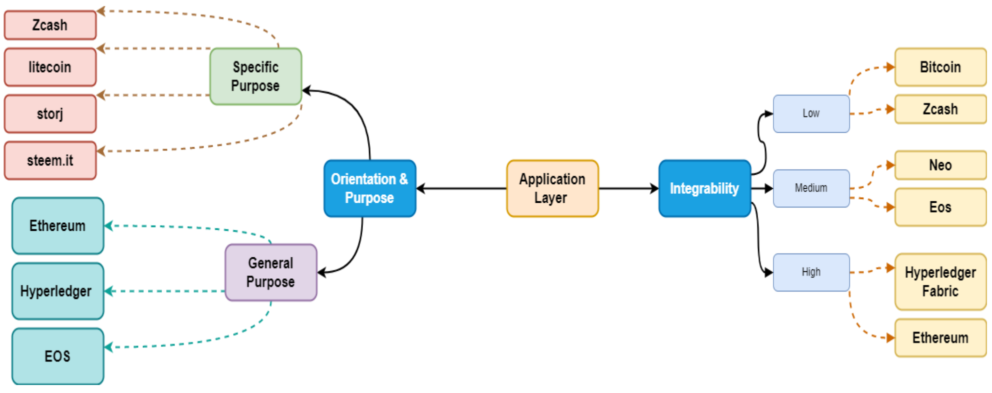

Fig. 4. The main components of the application layer with examples

systems (e.g. Web, mobile). The integrability of a DLT can be considered as a qualitative property, thus it is possible to deduce a ”Level of Integrability”. That is, we establish a small integrability scale from “high” to “low”.

DApp orientation and DLT’s purpose Decentralized software applications (or DApps for short) are software applications whose server and client tiers are decentralized and operating autonomously using a DLT. We consider a DLT as DApp-oriented if it focuses on offering the necessary tools for building and maintaining decentralized applications, using different protocols and APIs. Wallets are an important component of the application layer. Generally, they manage the user’s cryptographic identities. Wallets are responsible for all cryptographic operations related to the creation or storage of the user’s keys or digital certificates as well as the management of transactions.

6.2 Application layer: state of the art

Due to the vastness of different approaches and tools provided by different DLTs at the application layer, we overview only the application layer of few notorious DLTs.

Integrability DLTs generally introduce a layer of integration between external entities and their data and execution layer. DLTs like Ethereum NEO or EOS and others, have a richer toolset and integration tools. Ethereum offers a robust and lightweight JSON-RPC API with a good support for the JavaScript language. It provides Web3.js, an official feature-rich JavaScript library for interacting with Ethereum compatible nodes over JSON-RPC. Further, for a better integration into legacy systems, Camel-web3j connector provides an easy way to use the capabilities offered by web3j from Apache Camel DSL. In addition, Infura provides online access for external actors to communicate with the Ethereum chain, through Metamask , dropping the need for running an Ethereum node or client making the DApp easier for the end-user. Similarly, EOS presents a wide set of tools and features, easing its integration and interaction with external systems. In fact, EOS provides multiple APIs such as EOSIO RPC API, and its implementations in different languages (EosJs , Py Eos , Scala Eos wrapper , Eos Java , etc.). These tools enable developers to interact with EOS using most used programming platforms. B2B-targeting DLTs such as Hyperledger fabric or Corda platform, tackle the integrability issue by providing rich integration SDKs. For instance, Fabric provides a RESTful API Server which uses Fabric SDK as a library to communicate with the DLT network. Fabric SDK currently supports Node.js and Java languages. Other technologies like Bitcoin or its variants (Litecoin, Dogecoin, etc.) enable less integrability as it was not aimed, by design, to integrate or communicate with other systems. It provides basic RPC features along with unofficial implementation of its protocol in different languages such as BitcoinJ, limited python implementations (e.g. pybtc), and others.

DApp Orientation and DLT purpose Bitcoin and similar projects (e.g. Zcash, Litecoin) are created with the purpose to serve as mere secure digital cash networks. Thus, they are considered Cryptocurrency-oriented. Other DLTs try to propose along the cryptocurrency other types of P2P value transfers. In the case of storage oriented DLTs such as Sia Network, Storj, FileIo, Ipfs, the network manages data storage alongside a cryptocurrency. Similarly, the service-oriented DLTs propose services consuming the inherent token, such as "Steemit" which runs a social network or Namecoin which aims to provide a decentralized DNS. On the other hand, various DLTs are DApp-oriented and allow developers to build generic applications. For example, Ethereum, EOS, Stellar, TRON, and many others, propose a more flexible development environment for building DApps with built-in tokens. For more information about current blockchain DApps landscape we refer to this study [23].

7 The Distinction between Blockchain and Blockchain like system

When deconstructing DLTs system using the DCEA framework and evaluating the differences between the two high-level taxons blockchain and the blockchain-like, we observe that they share many common characteristics, as well as distinguishing properties (Table 2).

In a zoom-out view, we consider that a system is not a blockchain and belongs to the blockchain-like category, if it displays at least two of the following traits. First, lack of good decentralization. In fact, multiple DLTs consider sacrificing the decentralization or weaken it for different reasons such as better scalability, straightforward and seamless governance or because they are intended to be deployed in contexts that do not require decentralization. Second, a blockchain-like system tolerates data tampering and provides weak immutability either for states or transactions. Third, the data structure does not rely on chained blocks of transactions to store data. In Table 2, we summarize the distinctive settings of each category to enable the separation between the two categories of DTLs. However, both categories are not disjoint, but overlap—often considerably. We can find a blockchain-like system that exhibits all blockchain properties except one or two. Moreover, in our distinction, we do not rely on the operational settings —being public or not or being permissioned or permissionless— for the separation between blockchain and blockchain-like because a project deployed in public and permissionless settings (e.g. Ethereum) can be as well deployed in private and permissioned settings and vice-versa.

Table 2. Settings of blockchain AND blockchain-like in DCEA framework

	Components and properties	Blockchain	Blockchain-like
Data Layer	Data structure	Chain of block	Chainless model
	Shareability	Global	Restricted by design
	States management	On-chain	Off-chain
	Immutability	Strong	Weak
Consensus layer	Consensus identity model (membership)	Permissionless	Permissioned
	Governance	Democratic, Oligarchic	Dictatorship, Oligarchic
	Data ordering	Decentralized and open	Centralized or reserved
Execution layer	Conflict resolution	Longest-chain/ No-Forks	Longest-chain/No-Forks
	Turing completeness	Turing/Non-Turing complete	Turing and Non-Turing complete
	Openness	Closed/Oracle-based	Open/Oracle-based
	Interoperability	Non-interoperable/Interoperable	Non-interoperable/Interoperable
	Determinism	Deterministic	Non-deterministic
	Execution environment and rules enforcement	VM, Script runtimes	VM, Script runtimes
Application layer	Integrability	High, Medium, Low	High, Medium, Low
	DApp orientation	DApps, Cryptocurrency	DApps/ Cryptocurrency
	Wallet management	Built-in	Built-in or External

8 CONCLUSION

In this paper, we have proposed a comprehensive and referential framework to ease the understanding and the investigation of different approaches adopted by different DLTs at the four layers: data structure, execution, consensus and application layers. We have defined a stack of DLTs components and their main properties after analysing the design choices adopted by a large spectrum of existing DLTs solutions. The layer-wise approach adopted by DCEA is aligned with the DLT's modular architecture and will help to provide a better and modular understanding of DLTs to decision-makers, who could then make granular decisions at each layer to construct the best solution. Moreover, DCEA will serve as a comparative baseline to build a comparative analysis between different DLT variants. In the future work, we aim to apply this referential framework to classify existing DLTs into two broad taxa: blockchain and blockchain-like systems.

References

1. Ittai Abraham and Dahlia Malkhi. The blockchain consensus layer and BFT. *Bulletin of EATCS*, 3(123), 2017.
2. Aeternity. æternity - a blockchain for scalable, secure and decentralized æpps.
3. IEEE Standards Association. IEEE blockchain standards.
4. Nicola Atzei, Massimo Bartoletti, Stefano Lande, Nobuko Yoshida, and Roberto Zunino. Developing secure bitcoin contracts with BitML. In *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1124–1128, New York, New York, USA, 8 2019. Association for Computing Machinery, Inc.
5. Sloane Brakeville and Perepa Bhargav. Blockchain basics: Glossary and use cases, 2016.
6. Emily N Dawson, Athan Taylor, and Yvonne Chen. ISO/TC 307 Blockchain and distributed ledger technologies.
7. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 4 1988.
8. GlobalPlatform and Inc. GlobalPlatform Security Task Force Root of Trust Definitions and Requirements. Technical report, 2017.

9. ISO/TR. ISO/TR 23455:2019 Blockchain and distributed ledger technologies — Overview of and interactions between smart contracts in blockchain and distributed ledger technology systems.
10. ITU. Focus Group on Application of Distributed Ledger Technology.
11. IVY. GitHub - ivy-lang/ivy-bitcoin: A high-level language and IDE for writing Bitcoin smart contracts.
12. Aggelos Kiayias and Giorgos Panagiotakos. On trees, chains and fast transactions in the blockchain. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11368 LNCS, pages 327–351. Springer Verlag, 2019.
13. Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.
14. Maker. Maker - Feeds price feed oracles.
15. Trent McConaghy, Rodolphe Marques, Andreas Müller, Dimitri De Jonghe, Troy McConaghy, Greg McMullen, Ryan Henderson, Sylvain Bellemare, and Alberto Granzotto. Bigchaindb: a scalable blockchain database. *white paper, BigChainDB*, 2016.
16. Nova Mining. Rootstock (RSK): Smart contracts on Bitcoin. Medium, 2018.
17. Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. {CHAINIAC}: Proactive software-update transparency via collectively signed skipchains and verified builds. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1271–1287, 2017.
18. Jack Peterson, Joseph Krug, Micah Zoltu, Austin K Williams, and Stephanie Alexander. Augur: a Decentralized Oracle and Prediction Market Platform. Technical report, 2018.
19. Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.
20. Randaow. GitHub - randao/randao: RANDAO: A DAO working as RNG of Ethereum.
21. Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.
22. F Tschorsch and B Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys*, 2015.
23. Kaidong Wu. An Empirical Study of Blockchain-based Decentralized Applications. *arXiv preprint arXiv:1902.04969*, 2019.
24. Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town Crier: An Authenticated Data Feed for Smart Contracts. *dl.acm.org*, 24-28-Octo:270–282, 10 2016.