|  |  |  |
| --- | --- | --- |
| ITU logo | INTERNATIONAL TELECOMMUNICATION UNION**TELECOMMUNICATIONSTANDARDIZATION SECTOR**STUDY PERIOD 2017-2020 | FG-AI4H-F-025-A03 |
| **ITU-T Focus Group on AI for Health** |
| **Original: English** |
| **WG(s):** | Plenary | Zanzibar, 3-5 September 2019 |
| **DOCUMENT** |
| **Source:** | FG-DLT |
| **Title:** | Updated baseline text: D3.1 - Distributed ledger technology reference architecture |
| **Purpose:** | Discussion |
| **Contact:** | Wei KaiCAICTChina | Email: weikai@caict.ac.cn |
| **Contact:** | Zhang YihuiCAICTChina | Email: zhangyihui@catr.cn |
| **Contact:** | Yang BaixueCAICTChina | Email: yangbaixue@caict.ac.cn |
| **Contact:** | Qing SudeCAICT China | Email: qingsude@caict.ac.cn |
| **Contact:** | Hu Ning OnchainChina | Email: hehehu@gmail.com |

|  |  |
| --- | --- |
| **Keywords:** | Blockchain, Distributed Ledger, Distributed Ledger Architecture |
| **Abstract:** | This document contains the updates made on draft ITU-T FG DLT deliverable D3.1 “Distributed Ledger Technology Reference Architecture” during the Madrid meeting. |

**ITU-T FG DLT deliverable D3.1**

Distributed ledger technology reference architecture

Table of Contents

[1 Scope 4](#_Toc5201925)

[2 References 4](#_Toc5201926)

[3 Terms and Definitions 4](#_Toc5201927)

[3.1 cross-chain interoperability 4](#_Toc5201928)

[3.2 event model 5](#_Toc5201929)

[3.3 state model 5](#_Toc5201930)

[3.4 UTXO model 5](#_Toc5201931)

[3.5 balance model 5](#_Toc5201932)

[3.6 address 5](#_Toc5201933)

[3.7 identity 5](#_Toc5201934)

[4 Disclaimers 5](#_Toc5201935)

[5 Abbreviations and acronyms 5](#_Toc5201936)

[6 Architecture Overview 6](#_Toc5201937)

[6.1 Overview 6](#_Toc5201938)

[6.2 Resource & infrastructure functions 7](#_Toc5201939)

[6.3 Protocol / governance & compliance functions 7](#_Toc5201940)

[6.4 DApp application(s) functions 8](#_Toc5201941)

[6.5 Operation & maintenance functions 9](#_Toc5201942)

[6.6 External interaction management functions 9](#_Toc5201943)

[6.7 Extension functions 9](#_Toc5201944)

[7 Functional Components 9](#_Toc5201945)

[7.1 Core layer 10](#_Toc5201946)

[7.1.1 Network and infrastructure 11](#_Toc5201947)

[7.1.2 Extendable protocol communication 11](#_Toc5201948)

[7.1.3 Network (P2P Network) management 11](#_Toc5201949)

[7.1.4 Storage service 11](#_Toc5201950)

[7.1.5 Consensus mechanism 12](#_Toc5201951)

[7.1.6 Smart contract mechanism 13](#_Toc5201952)

[7.1.7 Ledger management 15](#_Toc5201953)

[7.1.8 Data protection 16](#_Toc5201954)

[7.1.9 Utility 16](#_Toc5201955)

[7.2 Service layer 16](#_Toc5201956)

[7.2.1 Account management 16](#_Toc5201957)

[7.2.2 Node (System) management 17](#_Toc5201958)

[7.2.3 Smart contract mechanism 18](#_Toc5201959)

[7.2.4 Data protection 18](#_Toc5201960)

[7.2.5 Trust endorsement management 18](#_Toc5201961)

[7.3 Application service platform 18](#_Toc5201962)

[7.3.1 DApp framework 19](#_Toc5201963)

[7.3.2 Accounting, authorization and authentication 19](#_Toc5201964)

[7.3.3 Data privacy 19](#_Toc5201965)

[7.3.4 Data storage & synchronization 20](#_Toc5201966)

[7.3.5 Operation & maintenance 20](#_Toc5201967)

[7.4 DLT applications 20](#_Toc5201968)

[7.5 External services 21](#_Toc5201969)

[7.5.1 Offchain system(s) 21](#_Toc5201970)

[7.5.2 External interaction/interoperation management 21](#_Toc5201971)

[7.5.3 External resource management 21](#_Toc5201972)

[8 Architecture mapping of other distributed ledgers 21](#_Toc5201973)

[9 Comparison with Architecture with ISO TC307 21](#_Toc5201974)

[10 Bibliography 22](#_Toc5201975)

**ITU-T FG DLT deliverable D3.1**

Distributed ledger technology reference architecture

Summary

This document defines the reference architecture for distributed ledger technology, the hierarchical relationship and specific functions of the distributed ledger technology architecture, important modules and specific functions in the structure of distributed ledger technology, the main technical route and direction of the core module in the distributed ledger architecture.

Keywords

Distributed Ledger, Blockchain, Reference Architecture

Introduction

# Scope

This document defines the reference architecture for distributed ledger technology, the hierarchical relationship and specific functions of the distributed ledger technology architecture, important modules and specific functions in the structure of distributed ledger technology, the main technical route and direction of the core module in the distributed ledger architecture.

This document can be used as a guideline for DLT service providers to build system, and for the organizations to select and use a DLT system.

# References

The following ITU-T Recommendations and references referred in the text in such a way that some or all of their content constitute provisions of this document.

* ITU-T FG DLT Deliverable D1.1- DLT terms and definitions

 *[to be added]*

# Terms and Definitions

This document applies the terms and definitions defined in ITU-T FG DLT Deliverable D1.1.

## cross-chain interoperability

a) The ability of two or more blockchain systems to exchange information and use each other's information

b) The ability of two or more blockchain systems to operate with one another

## event model

## state model

## UTXO model

## balance model

## address

## identity

# Disclaimers

(1) Sample projects and reference articles mentioned in this deliverable are only for the purpose of analysis of technical architecture. The Focus Group does not endorse any of these projects and/or reference articles, nor their technical aspects.

(2) The editors have included these sample projects and/or reference articles based on the availability of mapping documents in the appendix[[1]](#footnote-1) as examples for better explaining DLT. The inclusion of these examples does not imply any endorsement of, or judgement on, the quality or applicability of the mentioned implementations of the technology.

# Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

PoW Proof of Work

BFT Byzantine Fault Tolerance Algorithm

PoS Proof of Stake

DApp Decentralized Application

DPoS Delegated Proof of Stake

DLT Distributed Ledger Technology

dBFT Delegated Byzantine Fault Tolerance Algorithm

VBFT Byzantine Fault Tolerance with Verifiable Randomness

JVM Java Virtual Machine

EVM Ethereum Virtual Machine

IBE Identity Based Encryption

PKI Public Key Infrastructure

SHA Secure Hash Algorithm

SM3 Shang Mi 3[[2]](#footnote-2)

DES Data Encryption Standard

AES Advanced Encryption Standard

ECC Elliptic Curves Cryptography

TXO Transaction Output

UTXO Unspent Transaction Output

KYC Know your customer

RPC Remote Procedure Call

SDK Software Development Kit

# Architecture Overview

## Overview

The highlevel architecture constrains the highly abstract hierarchical architecture of distributed ledgers. The top-level architecture can cover almost all distributed ledgers, including public chains represented by Ethereum[b-ethe] and Bitcoin[b-bitc], private chains represented by Hyperledger Fabric, and non-blockchain distributed ledgers systems[[3]](#footnote-3).



Figure-1 High Level Conceptual Architecture of DLT

## Resource & infrastructure functions

The Infrastructure provides the operating environment and basic components required for the normal operation of the distributed ledger system. The base layer includes network services, storage services, and computing services. The basic layer is the resource that most software systems rely on and is the foundational support of the distributed ledger system.

##### Network management functions

Each distributed ledger instance is built upon a network hypothesis, which outcomes the distribute model of the system. E.g., in the study of blockchain, in a decentralized network, that each node inside blockchain network has got the same privileges, thus P2P network model is used.

##### Storage management functions

Each distributed ledger instance has got a standard storage component to save data, and ensure data protection. Esp., in the study of blockchain technique, based on the cost of distributed storage, storage management is to provide solutions for on-chain business to balance cost and data protection.

##### Utility functions

Distributed ledger technique has got typical utility functions to protect data, not only raw data protection, but also data transfer.

##### Node management functions

Each node inside the network of distributed ledger instance is maintained by node owner/operators. Node management is a component to manage the resource of a single node inside DLT instance.

## Protocol / governance & compliance functions

In DLT system, esp., blockchain systems, each node can have its own implementations based on system technical specification. Protocol is a conceptual layer to serve the technical specification cross nodes inside DLT system. The protocol layer includes “governance (and compliance)”, “consensus”, “ledger management” and “messaging”. Furthermore, the “governance and compliance” includes “node management”, “AAA management” (“Account management” and “right management”). Its function is to support the management of system governance (based on trust endorsement hypothesis) and AAA functions of other components.

##### Consensus mechanism functions

A consensus mechanism is the core component of the distributed ledger, esp., decentralized ledger, and is used to ensure the consensus of all nodes on the data. The consensus mechanism contains data consistency algorithms (aka., consensus algorithms), data validation, data distribution and synchronization. By use of the consensus mechanism, the distributed ledger system setup a trust mechanism upon the network hypothesis. Trust endorsement module, e.g., incentive mechanisms, is built upon that. Consensus mechanisms can be used to maintain public chain data, and can also maintain data based on various distributed ledger partitioning mechanisms.

##### Ledger management functions

The ledger provides basic data management of distributed ledgers and distribution management of ledger data on the network. It defines the local data storage methods of ledgers and the synchronization mechanism between nodes and responds to rights management, by use of consensus mechanism.

## DApp application(s) functions

Based on runtime management, DApp is built to serve different business in distributed network environment.

Furthermore, web applications with a combination of both off-chain services and on-chain services is a solution for real business.

##### Smart contract mechanism functions

With the development of blockchain technology, blockchain has been extended from simple accounting transactions to support complex transactions. Such complex transactions are stored in blockchain network in the form of bytecode programs, as an extension of distributed ledger module. According to the blockchain terminology, the bytecode programs are called smart contracts.

To support smart contract, smart contract mechanism carries specific services on the blockchain, including the language definition, compilation, and execution of the code. Smart contracts for different block chains can be implemented using simple interpreted scripts or fully functional virtual machines.

Smart contract mechanism is an optional component for DLT.

##### DApp management functions

An open DApp management layer is a middleware, to separate the distribute ledger network with the DApp business.

The DApp management middleware contains a DApp framework for DApp developers to create and maintenance DApps, and a smart contract management mechanism for DApp hosts to manage their DApps easily.

Furthermore, DApp framework can provide a series of interfaces for DApp developers. The interface provides the access to use of distributed ledgers. DApp users access the blockchain services through the interface. The interface can usually have API, SDK, RPC, and so on.

## Operation & maintenance functions

The operation and maintenance functions include various libraries such as log, monitoring, node/network management, and scaling libraries.

## External interaction management functions

Each distributed ledger instance has got its own network hypothesis, trust endorsement hypothesis and governance model, business is more complex. Thus, a distributed ledger instance with open-network hypothesis is able to interact/interoperate with external system(s).

In most cases, external interaction management is the runtime engine of external resource management.

## Extension functions

Extension component of DLT system is targeting to resolve different requirement of data interoperability. “Extension functions” includes a series of protocols/specifications for data interoperations of external systems, e.g., “multi-chain”, “side-chain”, “off-chain”, or internal systems, e.g., “child-chain”, “sharding [b-sharding-1] [b-sharding-2]”.

##### Internal system extensions

Each blockchain system has one governance model. Internal system extension is targeting to resolve the problem of scalability for one blockchain ecosystem, with the same governance model.

###### Child-chain

Child chains are individual ledgers with their own native tokens responsible for operational transactions such as deploying smart contracts, issuing assets, voting on polls, and sending messages. All child chains receive consensus from, and share the same source code as, the network’s main (“parent”) chain; therefore, all child chains on the network are interoperable.

##### External system extensions

In most business cases, the data interoperability is a cross system requirement. A blockchain system shall be able to access external systems to satisfy business requirements.

###### Off-chain system(s) functions

In the study of blockchain, to interact/interoperate with off-chain systems are scenarios in most use cases.

Furthermore, a series of L2 (layer 2) solutions, a combination of on-chain and off-chain technique, are used to optimize the performance, as well as the scalability, of blockchain system.

# Functional Components

The detailed architecture does some parse on the top-level architecture. Although the different distributed ledger platforms are highly consistent on the top-level architecture, the components in the detailed architecture are different. The next part will explain in detail the components and functions of the detailed architecture.

 

 Figure-2 Architecture diagram

This part mainly explains the significance and role of each module of the detailed architecture.

## Core layer

Functions map to “resources” and “Protocol”.



Figure-3 Typical flow of blockchain

A typical DLT system is targeting to execute transactions (events) and archive the result (state) in a distributed system. The standard process of this,

1. Event, to gather event from client(s)
2. VM, to prepare the environment for event execution
3. Execute, to execute the transaction(s) inside the event, get state result. The state result stands for the status of ledger, supports different state models (UTXO, balance)
4. Store, to store data (state, and/or event) into database(s)

In decentralized system, a consensus mechanism is required, to solve data consistency in different nodes. There’re two modes in blockchain.

State mode, consensus upon states, mostly used for pre-execution, UTXO models.

Event mode, consensus upon events, mostly used for post execution, balance models.

In cross-chain scenarios, state mode chain is usually used after chain consensus.

### Network and infrastructure

As an IT solution, DLT node works with typical distributed system solutions (including cloud solutions).

#### Safe hardware

DLT system works for data protection and data privacy with high performance, trusted/safe hardware can be optional, e.g., trust execution environment (TEE).

### Extendable protocol communication

Scalable protocol communication module based on network hypothesis cross distributed system.

The communication component is targeting to give the capability of data exchange cross different chain network, L7 filtering for homogeneous blockchain network if possible.

### Network (P2P Network) management

A DLT system is based on a network hypothesis, thus network management is required. Esp., in blockchain system, each node has got the same privileges, the node owner is able to decide the contribution of the node itself, network management is to provide basic capability to control node network.

#### Network discovery

In a distributed ledger system, there are usually many nodes, especially in public chain systems. Each node needs to discover neighbour nodes through a network discovery protocol and establishes a link with neighbour nodes. According to different distributed ledger architectures, the network discovery protocol also needs to identify and authenticate the node identity to prevent various attacks such as sybil attack.

#### Data transceiver

After the node connects to the neighbour node through the network discovery protocol, data exchange (e.g. transaction broadcast, consensus message, data synchronization) with other nodes is completed by the data transceiver module. According to the architecture of different distributed ledger, the design of data transceiver needs to consider requirements such as serial number, confirmation, encryption.

### Storage service

#### Data persistence

Distributed ledgers need to save to persistent storage a variety of data, such as block data, transaction data, status data, and private data of a local account. Depending on the type of data and the design of the distributed ledger, different save modes can be used. Save modes include relational database such as MySQL[b-mysql], non-relational databases such as LevelDB[b-leveldb], and self-organizing files.

### Consensus mechanism

#### Data synchronization

The data synchronization module ensures that the distributed ledger has a consistent ledger. The data synchronization module transmits the new part of the ledger between different nodes. The synchronization module also validates the synchronized data to ensure the correctness and consistency of the synchronized data.

Distributed books of different architectures or different types of book nodes have different synchronization methods. Different synchronization methods include synchronizing all transactions and blocks and status data, synchronizing all transactions and block data, synchronizing partial transaction data, and so on.

#### Legitimate validation

Consensus is the core of distributed ledger. The purpose of consensus is to make many nodes involved in accounting jointly maintain a consistent account. It should be noted that the consensus algorithm here refers only to algorithms that agree on the transaction order, and noverification of the transaction data itself.

Consensus algorithm design should ensure the following:

**Consistency:** Consensus nodes eventually need to agree on the data.

**Timeliness:** Consensus nodes should complete the data consensus in as short a time as possible.

**Security:** It takes a huge cost to undermine consistency and cannot be easily attacked.

In theory, all kinds of algorithms can meet the above requirements and be applied in a certain scene as the consensus algorithm of distributed ledger.

#### Consensus algorithm

**PoW** (Proof of Work[b-pow]), commonly known as mining. PoW can be simply understood as providing a proof that you have done a certain amount of work. PoW requires the node to carry out a certain amount of computation to obtain the accounting right, which means that it takes a certain amount of time to be consumed by the computer and calculates a verifiable result through the mathematical operation. The node sends the data that needs to be recorded in this round. After verification, the other nodes in the whole network store the data together.

The main feature of the PoW system is that the prover needs to do more work to get the result, while the verifier can easily verify whether the prover has done the corresponding work through the result. A core feature of this approach is asymmetry: work is more difficult for a prover and easier for approver (such as HASH algorithm).

**Series of Byzantine Fault Tolerance Algorithm (BFT):** Byzantine fault tolerance is a common solution to achieve efficient fault tolerance. The system comes from Byzantine general problem. The system based on Byzantine fault tolerance algorithm can reach consensus when the number of failure nodes or fraud nodes is less than the number of fault-tolerant nodes. Generally speaking, the number of fault-tolerant nodes is less than one-third of the total number of nodes.

At present, there are many implementations of BFT series algorithms. The typical implementation method is the state machine copy replica algorithm. Under the control of the master node, all nodes confirm the status of other participants through the three-phase protocol and determine the accounting according to the status of all the participants content. BFT series algorithms currently have synchronized PBFT algorithm[b-pbft], asynchronous Honey Badger BFT[b-hbft] algorithm, and open channel BFT algorithm etc. There are also some BFT algorithms that support dynamic changes in the number of nodes.

**PoS** (Proof of Stake[b-pos]), PoS allows the so-called ‘token holders’ to replace the miners, Accountant is the holder of relevant tokens, and accounting right is their ‘Stake’. A typical PoS reduces the difficulty of mining in equal proportion to the percentage and time tokens that each node occupies in order to find the ‘verifiable result’ and determine the ownership of accounting rights faster.

**DPoS** (Delegated Proof of Stake[b-dpos]), each token holder determines the accounting right of the blockchain by voting, similar to the election of the board of directors. All nodes whose votes exceed the agreed votes become system trustees, forming a “board of directors” and alternately signing blocks. If a director missed the chance to sign a block, the nodes would vote for the others. Those boards that miss the chance to sign are disqualified and others can join the board.

**Traditional consensus algorithms:** Traditional consensus algorithms are based on traditional distributed consensus techniques, including PAXOS[b-pxos], RAFT[b-raft], and others. There are many similarities between the traditional consensus algorithm and other the consensus algorithms, all of which are aimed at solving the data inconsistency caused by network failure, hardware failure and data loss in the system. The traditional consistency algorithm pays more attention to performance and has higher requirements on the environment. At present, the traditional consistency algorithm is also widely used in DLT.

**Hybrid Consensus Algorithm:** The current consensus algorithms have their strengths and weaknesses, so there are some cases where consensus algorithms are mixed. Including BFT-RAFT[b-brft], combined with the high performance of RAFT and support for fraud nodes; dBFT[b-dbft], authorized BFT algorithm, integrated BFT with election and authorization mechanism for more nodes scenes; VBFT[b-vbft], achieves chain scalability by consensus node selection with VRF, anti-attack ability by randomness and PoS, and fast state finality with BFT.

### Smart contract mechanism

Smart contracts are an extension of the function of distributed ledger, which are deployed dynamically on distributed ledgers and decentralized to complete business operations.

The design of smart contracts usually includes the contract engine, contract code management and contract data management.

Contract code management is responsible for the operation of the deployment and storage of the contract code.

Contract data management maintains contract data and provides access interface for the contract engine.

The contract engine can execute the contract code and maintain the contract data according to the contract code.

#### Verification

The module contains a transaction buffer pool for distributed ledgers and basic verification of transactions. Distributed ledgers typically verify the sender's balance of the transaction, the sender's transaction number, and so on. Once the transaction is legal, the transaction is saved in the blockchain. The verification module can effectively prevent various attacks and ensure the stable operation of the distributed ledger.

#### Language and compile

For very limited business requirements, you can use a script to implement the contract, which is fast, secure and low in resource consumption.

For lightweight businesses, an inline contract engine is recommended, which is highly efficient and reduces the complexity of deployment.

For heavyweight business, an external execution environment is recommended, which has more resources and higher execution efficiency, it compatible with the mainstream programming language and is easy to develop.

Language andcompile provides the grammar specification of the contract as well as the compilation specification to ensure that the transaction can be processed by the execution engine.

#### Execution engine

**External contract execution environment:** It uses the external execution environment to execute the contract code, and usually uses an external container (such as Docker[b-dock]) or virtual machine to ensure a consistent execution environment for all nodes. The maintenance of contract data is accomplished by the distributed ledger process communicating with the agent process in the external environment, typical representative includes the realization of the contract in the Docker container of Hyperledger.

The advantages of this contractual technology are Turing complete, easy-to-master programming languages and efficient contract development. The heavyweight business is more efficient due to the permanent contractual execution environment. The disadvantages are slow deployment of contract code, weak external engine attack resistance, large resource usage, large inter-process communication overhead, and not suitable for short contract codes.

**Inline contract engine:** A contract engine is embedded in the process, and the contract codes are interpreted during execution or implemented after being compiled. The engine can use open third-party engines such as JVM[b-jvm], LUA[b-lua], JS[b-js] and other scripts; or be implemented by yourself, such as EVM[b-evm]. At present, most distributed ledgers use this technology, e.g. Ethereum.

The advantages of inline contract engine are Turing complete, fast contract deployment, fast data access, high security of the engine, and convenient management and customization of the computing resources of the contract engine. The disadvantages are that the contract is generally not resident in memory, and the engine efficiency is difficult to compare with the external engine, making it less efficient to perform heavyweight business.

**Scripting/Finite State Machine:** A set of execution instructions is preset in the distributed ledger, this sequence forms a piece of contract code, which is interpreted by the state machine. Due to the complexity of the instructions and the size of the code, the script generally has a weak function and is not Turing complete, it only performs basic operations. Many current virtual currency ledgers, including bitcoin, use this technology.

Advantages, the technology is easy to implement, its script functions are effectively controlled, and a strong anti-attack ability. Disadvantages including complex script development, non-Turing complete, and the scale of scripts is small, only to complete very limited functions.

### Ledger management

The ledgers store all the transaction data and contract data. The ledgers need to be able to complete the transaction and contract’s processing, indexing and storage.

Ledger management contains two components,

1. Ledger mechanism, based on consensus mechanism, relying on P2P networking and extended storage services, event execution upon the consistent event/or, state update upon consistent state.
2. Trusted storage, extended storage services for ledger.

Consensus algorithm design should ensure the following:

The books provide a complete definition of the various types of data such as transactions, blockchain, assets, accounts, etc., and provide a comprehensive index of various data.

The transaction data in each block of the ledger can be summed up using an algorithm such as Merkle Tree, the blocks can be continuously added, and different blocks can be chain-linked to ensure that they cannot be tampered with.

The design of the ledger can be used for data synchronization and verification, and the ledgers should support large-scale data storage and high concurrency.

#### Transaction record

Account: The account can use public-private key generated by asymmetric encryption mechanism, or you can use ordinary account.

Assets: Assets can be expressed in the form of account and balance model, or in the form of TXO (e.g. UTXO[b-utxo]) model, or a combination of the two. Assets can also be defined in the contract data independent of the underlying assets on the ledger.

Ledger Storage: The ledger storage is usually implemented as a custom file or as an embedded database. The ledger store is divided into two parts, one part is the block data, contains the original transaction, the other part is the result of the transaction or contract execution, usually stored outside the block.

Block structure: block consists of the transactions, the previous block digest, a variety of relevant dataand its own block digest. The digest of the block is usually calculated from the transactions in the block, the digest of the previous block, and various other related data according to an algorithm such as the Merkle Tree.

#### Status

Most distributed ledgers, in addition to saving blocks and transaction data, also hold some data or results of the transaction execution, which can be called Status. Because of the consistency of the distributed ledger smart contract engine execution, the status of different nodes of the distributed ledger can be consistent. For a simple distributed ledger like Bitcoin, the status may contain only a list of unspent assets, and so on. For distributed ledgers like Ethereum, Hyperledger with Turing-complete intelligent contract engines, the status preserves a more complex data generated during virtual machine execution.

By maintaining status data, distributed ledgers can continue to process a series of complex and continuous transactions.

### Data protection

The blockchain system provides data protection capabilities for data reading and writing of ledger and the use of trusted storage consistent data.

### Utility

#### Cipher library

Distributed ledgers use a variety of cryptographic algorithms. The cryptographic algorithm library provides basic cryptographic algorithm support for each component, including various commonly used encoding algorithms, hash algorithms, signature algorithms, privacy protection algorithms, etc. The cryptographic algorithm library also provides functions such as maintenance and storage of secret keys.

#### Messaging

The message module provides message notification services between different components within the distributed ledger and between different nodes. For example, after a successful transaction, the customer usually needs to track the results of the execution of the transaction and even some records during the execution of the transaction. The message module can complete the generation, distribution, storage and other functions of the message to meet the needs of the distributed ledger.

## Service layer

Functions map to “protocol” and “governance”.

The core layer provides four basic services for upper applications, data protection, data processing management, data AAA management and infrastructure management. These four services constitute the middleware between blockchain core and DApps.

### Account management

Distributed account system serves all kinds of entities and data in DLT system. Account management component controls addresses and identities.

address is bound with trust data. Esp., in public chain projects, most trust data are usually digital asset.

identity is bound with entity.

Based on distributed identity and multi-dimensional authentication protocol, account management component provides the identity management of node operators, trust endorsement regulatory roles, DApp operators, end users and extended Internet of Things access entities.

Esp., the account management of node identity, usually using the following technologies.

* CA: the certificates are issued through centralized CA for various applications in the system, the identity and authority management are certified and confirmed by these certificates.
* IBE: the identities are confirmed by IBE.
* PKI: the identities are confirmed by addresses/accounts based on PKI.
* Third party identity authentication: the identities are confirmed by the third party.

#### Distributed identity

Distributed identity establishes a cryptographic-based digital identity for business entities in DLT system. The digital identity is based on blockchain technology and is not subject to any centralized organization. It is completely controlled by business entities and is secure and trustworthy.

#### Authentication & Authorization

AAA (accounting, authentication and authorization) solution supports the full use of account management.

Authority managements have more differences since a variety of distributed ledgers have different application scenarios.

The existing business expansions can interface with the existing authentication and authority management.

The more concentrated authority management business can be complemented by using CA/IBE.

For instance, the public chains usually adapt PKI technology rather than the central authority management like CA/IBE.

#### Delegation

In DLT account management, non-user entities are also using distributed identities. The use of non-user entities shall enable id delegation, the entity owners can make full use of them.

### Node (System) management

System management includes secure communication, trusted data transmission, and related services of node operation and network governance.

#### System configuration

System configuration for node inside DLT system, includes network configuration, communication management configuration, DLT system global configuration, etc.

#### Node management

Node management for node inside DLT system.

#### Governance control

Governance control for node, and governance control to DLT system via multiple nodes, includes network status, communication channel monitoring, alarm and tracking, trust endorsement, node failure monitoring, etc.

#### Supervisory support

Component support for the monitoring and supervision of dishonest events in the blockchain network.

### Smart contract mechanism

Smart contract mechanism handles the trusted data processing, including smart contract lifecycle management, contract registration, pre-authorization, deployment, upgrading, iteration, cancellation. In service layer, smart contract mechanism provides the component for contract registration, contract template, contract compiler and VM runtime.

### Data protection

#### Policy configuration

The policy configuration of trusted data includes access templates, privacy templates, monitoring and auditing strategies, etc.

#### Access control

Trusted data access control, dynamic data operation control.

#### Data security and privacy management

Security management of data storage and privacy management for data affirmative easement.

#### Data monitoring and auditing

Tracking and monitoring of data, data processing, review and audit of special data events.

### Trust endorsement management

According to the different trust endorsement of distributed system, the trust endorsement management is used to meet the governance model. E.g., the legal endorsement of off-chain governance (consortium chains); the tokenomics endorsement of in-chain governance (public chains).

#### Incentive module

The incentive module is used for accounting incentives for partially distributed ledgers such as typical public chains. Most of the public chains use consensus algorithms for the targeted tokens distribution, and motivate the accounting nodes to account. In general, the incentive module and the consistency module are very closely related.

## Application service platform

Functions map to “application” and “operation & maintenance”.

### DApp framework

DApp framework includes interfaces which support DApp development, DLT data management and DLT account management.

The upper layer interface provided by the distributed ledger gives external systems an efficient access to the distributed ledger data, external applications to integrate distributed ledgers or other distributed ledgers for mutual access, usually includes RPC, API and SDK. The interface layer mainly completes data synchronization, transaction exchange, etc.

The RPC interface connects peripheral components with the distributed ledger nodes over the network and to access the services provided by the distributed ledger. SDK provides a development package for other components to integrate part of the functionality of a distributed ledger.

RPC and SDK should observe the following rules:

Complete functional: the transactions of distributed ledger can be completed and maintained, and an intervention strategy and privilege management.

Portable: it can be used in a variety of applications and environments, and is not limited to someone absolute software or hardware platform.

Extensible and compatible: it should be as forward and backward compatible as possible, and try not to modify or minimize changes as extending function.

Easy to use: the structured design and good naming methods should be used to reduce the cost of developement.

Common implementation techniques includes call control, serialized objects, and network components. There are various architectures which can be used, such as CORBA[b-corb], JsonRPC[b-jrpc], gRPC[b-grpc], Thrift[b-thrft], RestAPI[b-rapi], XMLRPC[b-xrpc] etc.

### Accounting, authorization and authentication

The AAA system manages the DApp users to access data, process data and do data exchange based on transactions.

The AAA management shall focus on three parts,

1. Access control for DApp users to submit their transaction
2. Access control for nodes to access the chain network (aka, permission control), usually being applied by private chains and consortium chain (aka, permissioned chain)
3. Privilege control for DApp users to operate the data and function calls via transactions

### Data privacy

The use of cryptographic technique to protect on-chain user data, to meet data privacy requirement for business applications (result in DApps).

Privacy has always been one of the obstacles to the application of distributed ledger, how to satisfy regulatory requirements and not infringe data privacy is the key to the distributed ledger industry.

Privacy protection should meet the following requirements:

1. Privacy, to ensure that users can set the transaction so it’s not visible to unrelated parties .
2. High-performance, privacy-protected design must be able to meet performance requirements.
3. Transparent supervision, privacy protection should not evade the regulatory functions of the regulatory agency.

### Data storage & synchronization

Useful tools for end users and DApps to do DApp data storage, synchronization, operation and credentials.

### Operation & maintenance

#### Deployment

The deployment of distributed ledger refers to the installation and use of distributed ledger services for different scenarios and users with different node permissions and service modes. According to the type, distributed ledger divide into public chain, consortium chain, private chain, their deployment methods are not the same.

Public chain: Public chain generally do not make any restrictions to nodes access, and less demanding on the operating environment, the ledger nodes are relatively simple, all users are free to participate in consensus and read and write data. Public chains distribute in the form of application market, freely installed and used by users.

Consortium Chain / Private Chain: In such distributed ledgers, consensus processes can only be involved with authorized customer nodes. Authorized nodes can participate in the consensus and data read and write process according to the rules. Consortium chain/private chain generally need to provide higher performance, so the operating environment requirements for consensus nodes are higher. It is recommended that such distributed ledgers use a high performance and consistent execution environment for deployment for higher performance and reliability.

The deployment of consortium chain / private chain can adopt the following ways:

Process-based deployment: Operations staff personnel deploy nodes on the host or virtual machine to complete the configuration. This deployment is flexible. However, due to the complex configuration of the configuration file, the deployment is inefficient and prone to problems caused by inconsistent node environment.

Container-based deployment: Ensuring a uniform environment within the container and encapsulating the distributed ledger in the container, then deploying the container by the operations staff. This approach is simpler to deploy than process-based deployment and more reliable.

Based on cloud service deployment: integration of distributed ledger and cloud services, rapid deployment of distributed ledger services through the cloud platform, while providing different levels of PaaS, BaaS services.

## DLT applications

Functions map to “application”.

Multiple applications based on DLT system, esp., blockchain system, DApps.

## External services

Functions map to “external interaction management” and “extensions”.

The reference architecture of DLT, esp., decentralized system (blockchain), shall meet the requirement to balance the needs of security, decentralization, and scalability. Furthermore, decentralized system is focusing on resolving "trust" issue in competitive business environment, that not all business shall use decentralized system.

A hybrid system combines DLT system, esp., decentralized system, with typical IT system can satisfy most business requirement.

From DLT system perspective, external services provide solutions to cooperate with external systems.

### Offchain system(s)

Offchain system includes non-DLT system, 3rd party DLT system and layer 2 blockchain technology.

Esp., layer 2 blockchain technology, its main purpose is to scale blockchain transaction capacity while retaining the decentralization benefits of a distributed protocol. Solving the scalability problem will significantly help with blockchain mainstream adoption. Layer 2 blockchain technology systems are those that connect to and rely on blockchain system as a base layer of security and finality.

Up to now, layer 2 solutions includes, plasma, state channel, sharding, raiden network, lightening network, etc.

### External interaction/interoperation management

In layer 2 solutions, blockchain system is able to interact/interoperate with layer 2 systems.

### External resource management

To cooperate with non-DLT systems and 3rd party DLT systems, mostly data/resource exchange transactions, resource management is required.

# Architecture mapping of other distributed ledgers

Ref., D3.2.

*[to be completed]*

# Comparison with Architecture with ISO TC307

As an international standard reference, it should be more general and suitable. The architecture reference provided by ITU-T FG DLT is a top-level framework, which could apply to different kinds of distributed ledger technology (DLT) systems. Compared with this architecture, the document of ISO-TC307 report, which faces to realization based on the user and node vision, maybe define a function framework referred to the DLT platform of Fabric and similar systems. Appendix I Architecture mapping of to existing DLT platform

[to be completed]

# Bibliography

[b-bitc] https://bitcoin.org/en/

[b-brft] http://www.scs.stanford.edu/17au-cs244b/labs/projects/clow\_jiang.pdf

[b-corb] http://www.corba.org/

[b-D1.1] ITU-T FG-DLT Base document D1.1 “Distributed ledger technology terms and definitions”

[b-dbft] http://docs.neo.org/en-us/basic/consensus/consensus.html

[b-dock] https://www.docker.com/

[b-dpos] https://bitshares.org/technology/delegated-proof-of-stake-consensus/

[b-ethe] https://www.ethereum.org/

[b-evm] https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List

[b-grpc] https://grpc.io/

[b-hbft] https://github.com/amiller/HoneyBadgerBFT

[b-jrpc] https://www.jsonrpc.org/

[b-js] https://www.javascript.com/

[b-jvm] https://www.javaworld.com/article/3272244/core-java/what-is-the-jvm-introducing-the-java-virtual-machine.html

[b-leveldb] http://leveldb.org/

[b-lua] https://www.lua.org/

[b-mysql] https://www.mysql.com/

[b-pos] PoS

[b-pxos] http://lamport.azurewebsites.net/pubs/lamport-paxos.pdf

[b-raft] https://raft.github.io/

[b-rapi] https://restfulapi.net/

[b-sharding-1] https://github.com/ethereum/wiki/wiki/Sharding-FAQs

[b-sharding-2] https://github.com/ontio/documentation/blob/master/sharding/ontology-sharding.pdf

[b-thrft] http://thrift.apache.org/

[b-utxo] https://bitcoin.org/en/glossary/unspent-transaction-output

[b-vbft] https://ontio.github.io/documentation/vbft\_intro\_en.html

[b-xrpc] http://xmlrpc.scripting.com/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. TBD, D3.2 mapping projects as appendix later. [↑](#footnote-ref-1)
2. One of the hash algorithms specified in ISO/IEC 10118-3 [↑](#footnote-ref-2)
3. See appendix, [TBD] D3.2 - Overview of existing platforms and mapping to distributed ledger technology reference architecture - Corda [↑](#footnote-ref-3)