TECHNICAL REPORT

ISO/TR 23455

First edition
2019-09

Blockchain and distributed ledger technologies — Overview of and interactions between smart contracts in blockchain and distributed ledger technology systems

Reference number
ISO/TR 23455:2019(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 307, *Blockchain and distributed ledger technologies*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Smart contracts, a synonym for automated applications on blockchain and distributed ledger technology-based (BC/DLT) systems, are an important development step from early stage, purely transaction oriented blockchains to more interactive technologies where the transactions on the blockchain or distributed ledger technology system are conditional on the terms of that application. According to the current working-definition of ISO/TC 307, WG1, Terminology, a smart contract is a

"computer program stored in a distributed ledger system wherein the outcome of any execution of the program is recorded on the distributed ledger".

In specific implementations of BC/DLT systems, such a program can vary from program code interpreted on single peers to (pre-)compiled programs recorded on the ledger to be executed on arbitrary virtual machines within the system (such as miners). It should be understood that the "effects" to be recorded on the distributed ledger will usually be the transaction that is the deterministic, predefined coded outcome from the smart contract code.

As the term smart contract in its original intention as created by Nick Szabo in 1994 had a different, mainly legally oriented (precise and legitimate) meaning, this has often caused confusion regarding "legally binding intentions": As this document discusses and describes smart contracts as a technology for BC/DLT automation in general, it is also important to understand that smart contracts may have a legal binding intention. Because of this, the legal binding application and structure of smart contracts also requires understanding of legal background, context and definitions.

This document mainly describes the aspects of automated software in a BC/DLT-system.

Blockchain and distributed ledger technologies — Overview of and interactions between smart contracts in blockchain and distributed ledger technology systems

1 Scope

This document provides an overview of smart contracts in BC/DLT systems; describing what smart contracts are and how they work. It also discusses methods of interaction between multiple smart contracts. This document focuses on technical aspects of smart contracts. Smart contracts for legally binding use and applications will only be briefly mentioned in this document.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
asset
anything that has value to a stakeholder

[SOURCE: ISO/TS 19299:2015, 3.3, modified — Note 1 to entry has been removed.]

3.2
ledger
information store that keeps records of *transactions* (3.10) that are intended to be final, definitive and immutable

3.3
miner
DLT node which engages in *mining* (3.4)

3.4
mining
block-building activity in some consensus mechanisms

Note 1 to entry: Participation in mining is often incentivized by block rewards and *transaction* (3.10) fees.

3.5
off-chain
related to a blockchain system, but located, performed or run outside a blockchain system

3.6
on-chain
located, performed or run inside a blockchain system

3.7
DLT oracle
oracle
distributed ledger technology oracle
service that updates a distributed *ledger* (3.2) using data from outside of a distributed ledger system

Note 1 to entry: *Smart contracts* (3.8) cannot access sources of data external to the distributed ledger system on their own; therefore, DLT oracles act as services designed to provide trustworthy data from external sources for use by a smart contract.

3.8
smart contract
computer program stored in a distributed *ledger* (3.2) system wherein the outcome of any execution of the program is recorded on the distributed ledger

Note 1 to entry: A smart contract might represent terms in a contract in law and create a legally enforceable obligation under the legislation of an applicable jurisdiction.

3.9
token
representation of a collection of data

Note 1 to entry: In this document, token is also used as synonym for a virtual asset (3.1).

[SOURCE: ISO/IEC 14776-323:2017, 3.1.85, modified — The original Note 1 to entry has been removed; a new Note 1 to entry has been added.]

3.10
transaction
smallest unit of a work process resulting in a state change

[SOURCE: ISO/TR 26122:2008, 3.5, modified — The words "consisting of an exchange between two or more participants or systems" have been replaced with "resulting in a state change".]

3.11
trust
relationship between two elements, a set of activities and a security policy in which element *x* trusts element *y* if and only if *x* has confidence that *y* will behave in a well defined way (with respect to the activities) that does not violate the given security policy

[SOURCE: ISO/IEC 13888-1:2009, 3.59, modified — Note 1 to entry has been removed.]

4 Symbols and abbreviated terms

BC/DLT: blockchain and distributed ledger technology

DSL: domain specific language

5 Overview of smart contracts

5.1 History of smart contracts

The term "smart contract" was first introduced by Nick Szabo in the early 1990s. But it was only with the advent of blockchain and distributed ledger technology (BC/DLT) that this concept gained widespread interest.

According to Szabo, a smart contract represents the idea of automatically fulfilling *contractual clauses* by embedding these clauses in a digital entity that has control over the property dealt with. This should be done “in such a way as to make breach of contract expensive […] for the breacher” (Szabo, 1997). Szabo therefore proposed the use of secure, machine-executable transaction protocols that ensure

automatic performance of predefined, conditional actions in accordance with the contract clauses. He saw smart contracts as a promising opportunity to significantly reduce “mental and computational transaction costs”. In his seminal paper, Szabo also described various well-known cryptographic techniques suitable for ensuring the desired characteristics of a smart contract, such as security, confidentiality and unforgeability. At that time, however, technology and market demand were not yet ready to implement these requirements comprehensively.

BC/DLT now provides capabilities that not only satisfy Szabo's demands but also offer advanced possibilities that reach beyond Szabo’s ideas. Massively distributed by design and safeguarded by a variety of cryptographic instruments (for example, transactions with pseudonymized accounts, immutability, unknown numbers of parallel executed and evaluated transactions), this environment allows smart contracts to be written in full-fledged programming languages, to communicate and interact with each other as well as with external resources, and to transparently keep track of their current state of execution. Thus, smart contracts with any relation to a legal context and its automation are actually a subgroup of all smart contracts as used for process automation on blockchains (see Figure 1). Examples for smart contract use-cases are listed in Annex C, Table C.1.

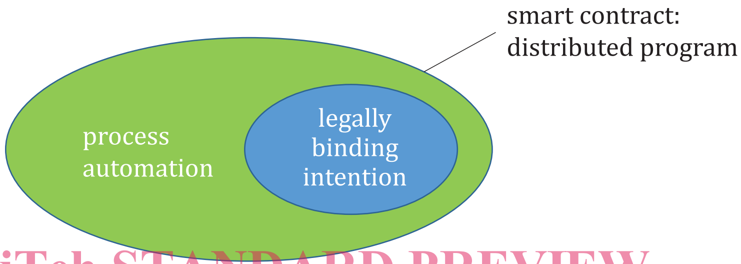


Figure 1 — Different understandings of the scope of “smart contract”.

Therefore, the smart contract concept is not limited to BC/DLT systems, smart contracts may also be used on traditional platforms (for example procurement portals); this document only considers smart contracts in the context of BC/DLT systems.

5.2 Different ways of understanding smart contracts

In the course of these technological developments, the understanding of the term “smart contract” has also evolved from its original meaning. Unfortunately, there is currently a lack of uniform understanding of the term "smart contract" in practice.

It is important to note that the term "smart contract" does not necessarily refer to a contract in the legal sense. Smart contracts can rather be taken to mean distributed applications that automate transactions by leveraging the security of DLT systems, and no implicit legal meaning should be inferred.

As already explained above, in 1996 Nick Szabo described smart contracts as being:

A set of promises, specified in digital form, including protocols within which the parties perform on the other promises[9].

In 2014, Vitalik Buterin[10] invented a new generation of smart contracts: decentralised and immutable once it exists in DLT systems. The notion of smart contracts within DLT systems is mainly developed on this new generation. For the purposes of this analysis, it is useful to consider three ways to understand smart contracts.

— **Piece of code:** According Szabo, he did not originally want to automate contracts, but instead to automate contractual clauses or exchanges:

  The basic idea behind smart contracts is that many kinds of contractual clauses (such as collateral, bonding, delineation of property rights, etc.) can be embedded in the hardware and software[11].

So it was intended to automate the evidence and consequence of contractual agreements and not the full act behind contracts and contracting; the binding and, in the event of dispute, enforcement.

— **Code as Law:** “Code is [shall be] law” has been widely used and specifically promoted by Lessig[41]. This terminology is not precise or reasonable as it blurs the distinction between “law” being “the system of rules which a particular country recognizes and enforces” and “a rule defining correct procedure (as in the laws of a game, the laws of physics, etc.)”. A somewhat less contentious alternative is “the code is the (smart) contract”, in which case the automation needs to include all consequences and their enforcement

— **As defined by legal professionals**: Common position of the legislators and legal professionals is the following:

The legal character of a smart contract is that which a judge or the law decides to be. So if the result of a smart contract should be enforceable, it is better to apply contractual standards to smart contracts.

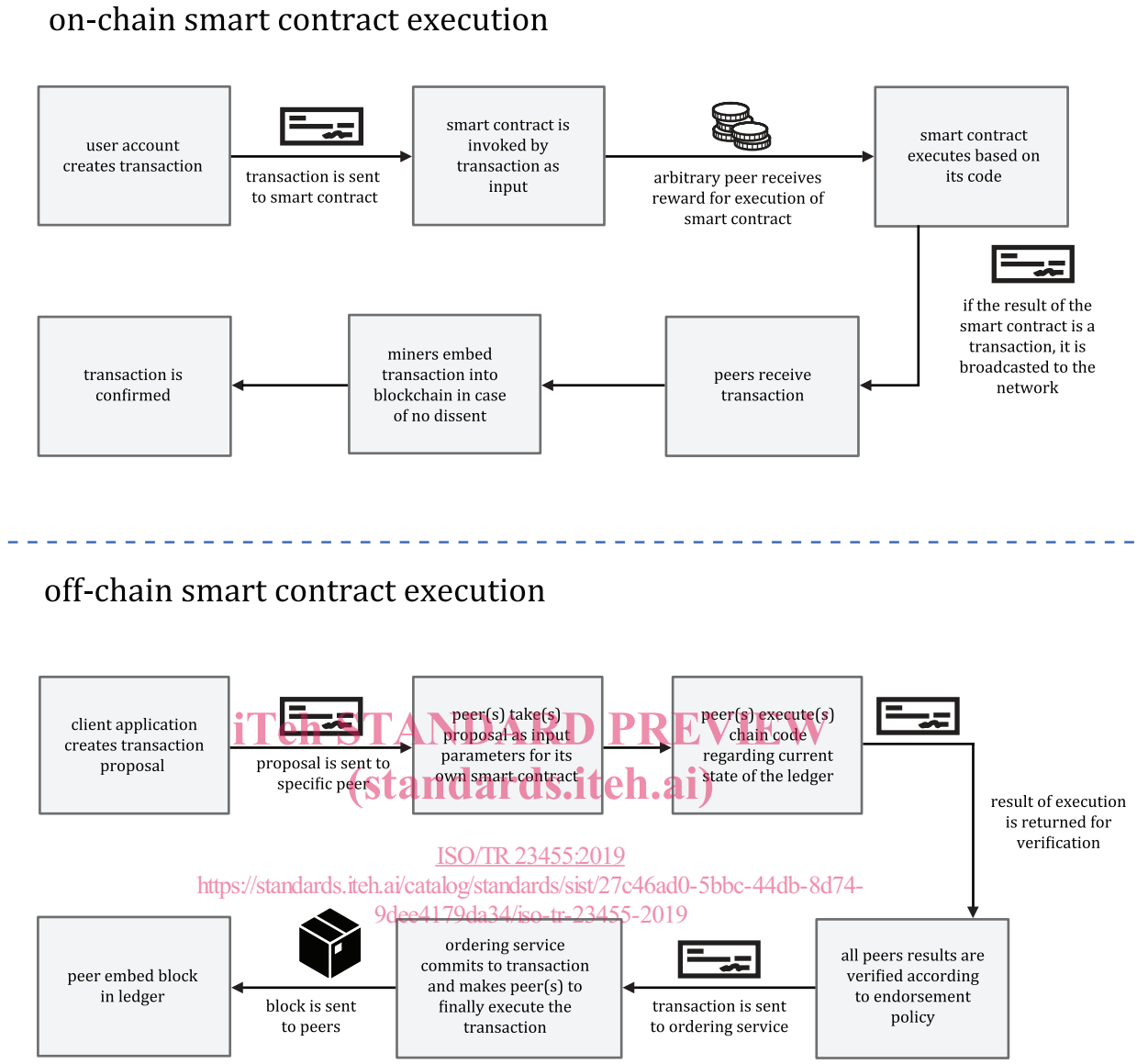
It is important to note that even a contract is not a statement of law; a contract is an agreement between parties that needs to comply with the requirements of applicable legislation including contract law.

6 Operation of smart contracts

6.1 The concept of a smart contract

DLT systems are designed to record transactions immutably into a ledger (examples for implementations are provided in Annex A). Some implementations support the deployment of custom code to the ledger, and later the invocation of deployed code with some parameters. Such deployed code is called a smart contract on a public DLT, and can itself interact with the underlying ledger such as user accounts or other smart contracts. It has to be mentioned, that the term “smart contract” is most popularly used in the context of public BC/DLT systems, for example Ethereum, in which the smart contract code may be executed with a deterministic result in an arbitrary virtual machine of a mining peer. Other concepts, for example on private DLT-systems such as Hyperledger Fabric, originally named the distributed code differently, for example “chaincode”, to emphasize differences in the execution concepts (see Figure 2). Meanwhile this term is also commonly referred to as smart contract. If there are any significant differences that require description, a precise distinction will be made between on-chain smart contracts and off-chain smart contracts.

Such code is installed and executed at selected peers. Consequently the deployment, execution and consensus process of such public and private systems differ.



NOTE See further explanation in 6.3.

Figure 2 — Difference between an on-chain and an off-chain smart contract execution

Thereby, smart contracts allow participants to implement and deploy custom functionality to augment the features of a running DLT system. Deployment and execution of smart contract code itself benefits from any transparency and immutability guarantees made by the principles and methods of the underlying DLT.

As well as generating changes to the underlying ledger state, some DLT implementations also allow smart contracts to control virtual assets (tokens). The clearest example of this is the case where smart contracts control an address, account or database item to which a cryptocurrency can be sent. The balance belonging to the address can only be sent by invoking the smart contract code itself, allowing a vast array of complex transactions to be performed under the security guarantees of the DLT system.

Similarly, the control of a smart contract, representing aspects of a legal contract between two parties can restrict handling and operation of the smart contract by checking the identity of potential users of a smart contract and the smart contract owner with the identity such as registered on the BC/DLT-system. This example also shows the benefit of using underlying security measures of the BC/DLT-system to increase the security of the control of a smart contract.

Once deployed, smart contracts are subject to the rules of the underlying BC/DLT platform. These implementation-specific rules do not necessarily allow smart contracts to be upgraded, modified or cancelled. Smart contract authors need to be aware of the implications of deploying to a live system, and carefully consider whether and how to build migration or termination features into their code. Here significant differences between private and public systems respectively off-chain and on-chain smart contracts may be observed: whereas on-chain smart contracts are usually operated and handled in a transaction-like manner, usually private off-chain smart contracts are part of a dedicated peer. Here off-chain smart contracts may be exchanged, upgraded or even manipulated without any significant notice to the remaining BC/DLT-system.

In order to control real-world assets via smart contracts, interactions with the outside need to be possible. For example, a smart contract's behaviour may depend on an external event or on information from external sources. In this case, trusted third party services or physical IoT-devices (called "DLT oracles") may be used to provide the data that triggers transactions in the DLT system (passing real-world information to smart contracts), or to monitor the ledger state and perform some action under specific conditions. Such DLT oracles may also be called by smart contract code.

6.2 Benefits and challenges of smart contracts

Even though they are generally considered a symbiotic alliance, smart contracts and distributed ledgers each are technologies on their own.

As an independent concept, smart contracts may reduce transaction cost in business relationships. As they formalize contractual terms or automation processes in clear computational logic, they significantly reduce room for misinterpretation and misunderstandings or accelerate processes. Furthermore, proven smart contracts for recurring tasks can be standardized and made available in libraries for re-use, which could be especially useful in government e-procurement from authorized suppliers. In this way, the contracting parties can quickly find consensus on implementations that best reflect their contractual intentions.

However, DLT systems provide an ideal environment to exploit the strengths of smart contracts. Instead of leaving it to the contracting parties to translate their individual interpretation of an agreement independently into proprietary business logic, the distributed ledger ensures that all parties involved see the same code at all times. Moreover, due to its consensus mechanisms, a distributed ledger provides each participant with unequivocal evidence of the occurrence of relevant events and the results obtained. Any replacement or update of a smart contract results in invalidating the old and replacing it with a new smart contract and needs to be validated by network participants under the terms of the consensus mechanism, so the network ensures full transparency and unforgeability of a smart contract and its current state of execution. As a result, distributed ledger-based smart contracts may reduce counterparty risk and transaction cost, and may make third party escrow services obsolete.

However, there are still some challenges with regard to distributed ledger-based smart contracts, with risks and benefits essentially stemming from the same characteristics. There is experience of malicious actors, also using smart contracts for various kind of profiling and discrimination or for involving the user into undesirable, unethical or illegal activities.

Of course, there are the well-known and widely discussed challenges regarding DLT systems in general, such as limited scalability, low performance, lack of privacy or danger of mining monopoly. Numerous projects are working on addressing these weaknesses.

However, there are also more specific risks and open questions that are closely related to the implementation of smart contracts on a distributed ledger. These will be elaborated in more detail in the following clauses. The following are some examples of typical problem areas.

From a conceptual perspective, most business use cases will require a smart contract to interact with the wider world, thus involving the support of trusted entities. This of course seriously questions the original idea of a fully decentralized system designed specifically to do away with third parties.

From a technical perspective, once switched on, a smart contract cannot be stopped from the outside as it is distributed to an unknown number of executing and evaluating parties. Unless the stop-mechanism

is actively programmed into the code, there is not a single "plug to pull" for stopping it. But there may be important reasons to halt the execution of a smart contract, for example in case of legal objections or a security breach. Moreover, as with all computer programs, a smart contract's behaviour depends on the input data, which makes it impossible to predict its behaviour or whether it will even terminate at all in all cases (halting problem). So what happens if a smart contract gets trapped in an infinite loop? Exhaustive algorithms can be limited in their execution by the requirement of paying for execution time. If the "execution currency" is consumed, the program automatically stops. Also, coding errors are unavoidable and their fault handling have to be covered.

From a legal perspective, smart contracts also raise many questions. For example, the "Code is law" dogma, while gaining popularity among programmers and IT professionals, is in conflict with many national law systems. It is still unclear e.g. whether a legal contract can be written in a programming language at all. If smart contract is considered as a tool, there is the question whether and when its outcome becomes legally binding. In addition, in the event of a dispute, the circumstances surrounding the conclusion of the contract also need to be taken into account when interpreting the contract.

6.3 Difference between on-chain and off-chain smart contracts regarding deployment and execution

NOTE The following text is also illustrated in Figure 2.

On-chain smart contracts usually operate in public BC/DLT systems following a three-step-process: deploy-invoke-operate.

— **Deployment**: During this phase a smart contract is made available to the blockchain system with a transaction containing the raw or (pre-)compiled code. In the deployment process this code-transaction obtains a smart contract address for later invocation or operation. The code itself is not active yet. Once being added to the distributed ledger, it is practically immutable, however, later on it may be cancelled or replaced with upgraded version just like a business offer.

— **Invocation**: In the invocation step the code is activated by sending a transaction to the ledger, calling its primarily assigned address, and potentially also transferring invocation parameters. During this step the code is loaded from the blockchain into the executing instances of the blockchain-system, for example the miners.

— **Operation**: After invocation the code is operated during the mining period usually in multiple distributed instances such as miners, consuming mining resources for operation which again and usually has to be compensated either by the smart contract itself or by the invoking instance. The result of the operation may be diverse, starting from a simple value to a new transaction up to another code deployment or invocation. After the operation the result is added to the ledger again in the typical consensus-process and made publicly available.

Usually on-chain smart contracts can also be disabled for execution by two mechanisms:

— If required for operation, the currency for compensating the operation cost on a miner can be withdrawn from a smart contract. This prevents it from being executed by miners. Recharging the smart contract with new compensation units enables the operation again.

— Additional stopping or disabling mechanisms are also usually provided. These prevent a smart contract at the original assigned address from being executed in its environment. Nevertheless, as the smart contract is recorded on the ledger, the pure binary code can be simply read from this transaction and be redeployed with a new transaction at a new address. This reactivates at least the functionality of the old smart contract.

Executing a smart contract off-chain may provide significant benefits compared with executing code natively on-chain, including:

— **Scalability**: Public distributed ledger networks are not currently suited to managing large throughput. This is problematic when running commercial smart contracts (particularly in large numbers) and where smart contracts share a network with other applications. Running smart

contracts off-chain provides scalability benefits as many smart contracts can be processed in parallel.

— **Privacy**: Privacy of the smart contracts is protected since computation occurs off-chain and transactions and/or state may be instantiated on-chain when required.

— **Security**: Exposing commercial terms on-chain increases the vulnerability of smart contract-based transactions.

— **Cost**: The computation cost of running smart contracts on-chain can become too high, particularly when execution cost is compensated with currencies coupled with the cryptocurrency itself.

Off-chain smart contracts such as used in many private BC/DLT systems are operated in the three-step-process: install-instantiate-invoke.

— **Install**: During this step a smart contract is made available to a peer in the blockchain system (no transaction is being written into the blockchain yet). The code itself is not active yet.

— **Instantiation**: In the instantiation step the code is activated on each peer of a channel (a peer has to be selected through an address; other channel members must endorse according to an endorsement policy) and instantiation parameters are (optionally) transferred.

— **Invocation**: After the code has been instantiated, it can be invoked by sending a transaction proposal to one or more peers. Endorsing peers must simulate and sign the transaction and send results to the ordering service. Afterwards, the invoking party verifies whether the results are valid (checks whether endorsement policy has been met), batches the transactions into blocks, orders them, and sends the transactions to the committing peers. The committing peers write the transaction into their local copy of the blockchain (ledger).

6.4 Access of real-world-information for smart contracts

6.4.1 General considerations about real-world-interaction

As smart contracts do not only operate and control blockchain-internal or user-triggered activities, but may also interact with real-world-information, an access to cyberphysical system needs to be provided. However, the interaction with the real-world is problematic as it provides a single point of access, making it vulnerable to

— false data,

— inconsistencies,

— security attacks.

As BC/DLT is associated with increased trust real-world data may unconsciously be associated with the same level of trust by the user.

The sources of real-world-data may be manifold and depend on the BC/DLT being used. Private systems using off-chain smart contracts directly attached to dedicated peers may operate local software directly accessing sensor data. Public systems with various randomly changing miners cannot attach a sensor to a miner but get their data as a transaction from the blockchain. This data needs to be sent from a DLT oracle, an external data creating entity, which is able to connect to a blockchain and to include the data into a transaction stored in the ledger. Besides physical sensors, such data creating entities can also be a website providing information such as weather services, news, aircraft arrival times or similar.

Faulty data can be delivered by external entities such as sensors or complete cyberphysical systems. The provision of such data cannot be prevented by the BC/DLT system or a smart contract but may be detected, for example by applying redundancy to the sensor system.

Inconsistencies may be caused by using multiple sensors for the same measurement which are delivering different values exceeding an expected range of deviation. This exceeded deviation may be caused by

various reasons such as: physical measurement deviations or scattering, time dependent measures causing different values for unsynchronized DLT oracles or data sources with counters creating access-depending indices never delivering the same data set of values over time.

Additionally sensors or DLT oracles are usually singular entities and therefore provide security leaks as single-points-of-attack. Consequently sensors may be replaced by others and be subject to false-data-injection, destruction or similar.

An overview of potential design criteria of an optimized sensor/DLT oracle/smart contract architecture is shown in Figure 3:

— Sensors are redundant and control and secure their values mutually by plausibility checks and deliver a harmonized value to the DLT oracle.

— To prevent security leaks by single-points-of-attack, DLT oracles are redundant. Additionally, they could monitor a spatially more extended sensor system from different local positions to avoid or detect inconsistencies and contradictory measurement. The redundant DLT oracle network delivers the data to the BC/DLT-system or the processing application, respectively.

— The smart contract relies on authoritative real-world data recorded in BC/DLT-system or is able to discover and to harmonize contradictory values itself.

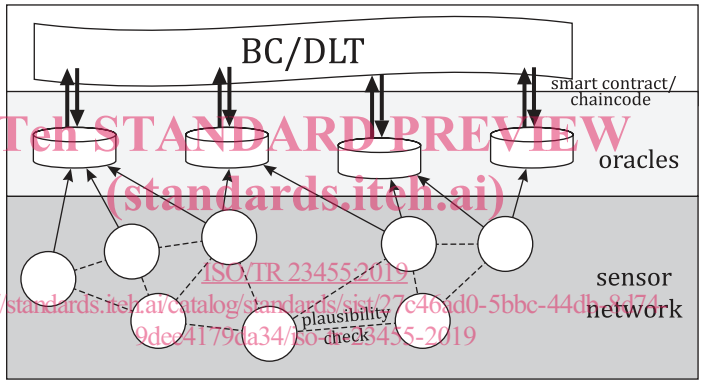


Figure 3 — Extended architecture for a more secure sensor connection to a BC/DLT-system via DLT oracles/interfaces

To increase trust in security and authenticity, DLTs generally require that all data is verified, in some sense. In the case of a transfer of value between two parties, the sender's address, the receiver's address and the amount to transfer are all signed by the sender before the transaction is transmitted to the network. Using only the data contained in the signed transaction it is straightforward for all nodes to verify that the controller of the sender's private key has authorized this operation on his or her data. All observers can then make this check and update the ledger in a deterministic way based on the data.

Digital signatures are an important tool in convincing validators of the veracity of real-world data. In particular multi-sig techniques can provide strong evidence, requiring that multiple parties sign (and thereby approve) any given value. Similarly, several DLT oracles could exist for the same data, allowing validators to consult multiple independent sources and ensure that they agree with each other. For the digital signatures to provide meaningful authenticity a secure binding of the real-world-identities to their signature keys is also needed, for example by means of a public key infrastructure (PKI).

6.4.2 One-way event delivery from a smart contract to an event consumer

In this scenario (see Figure 4), an event occurs within a smart contract, for example, a state change or the execution of a specific type of transaction. This event is broadcast to downstream consumers, and those consumers then take appropriate actions.