ETSI GS PDL 011 V2.1.1 (2022-09)

Permissioned Distributed Ledger (PDL); Specification of Requirements for Smart Contracts' architecture and security

Disclaimer

The present document has been produced and approved by the Permissioned Distributed Ledger (PDL) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
RGS/PDL-0011v211_SC_Arch_Sec

Keywords
PDL, policies, smart contract

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Permissioned Distributed Ledger (PDL).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document discusses the challenges and requirements of viable deployment of smart contracts for industries. The challenges due to inherent properties of smart contracts, and also due to external and internal interaction are discussed and their solutions are presented.

Introduction

The present document extends the discussion of challenges and requirements for the successful adoption of smart contracts. The present document discusses the current challenges of smart contracts' deployment and outlines architecture requirements that can mitigate those problems and enable error-free and efficient smart contracts. Moreover, the present document also oversees smart contracts' security aspects and explains internal and external threats to a smart contract and presents possible mitigation techniques for them.

1 Scope

The present document establishes the architectural and functional specifications of smart contracts, additionally, highlights the potential threats and specifies the solutions to mitigate them, requirements on the use of technology for smart contracts, governance, purpose, motivation and security.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI GR PDL 004: "Permissioned Distributed Ledgers (PDL); Smart Contracts; System Architecture and Functional Specification".

NOTE: Available at https://www.etsi.org/deliver/etsi_gr/PDL/001_099/004/01.01.01_60/gr_PDL004v010101p.pdf.

[i.2] ETSI GR PDL 010: "PDL Operations in Offline Mode".

NOTE: Available at https://www.etsi.org/deliver/etsi_gr/PDL/001_099/010/01.01.01_60/gr_PDL010v010101p.pdf.

[i.3] ETSI GS PDL 012: "Permissioned Distributed ledger (PDL); Reference Architecture".

NOTE: Available at https://www.etsi.org/deliver/etsi_gs/PDL/001_099/012/01.01.01_60/gs_PDL012v010101p.pdf.

[i.4] ETSI GR PDL 006: "Permissioned Distributed Ledger (PDL); Inter-Ledger interoperability".

NOTE: Available at https://www.etsi.org/deliver/etsi_gr/PDL/001_099/006/01.01.01_60/gr_PDL006v010101p.pdf.

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the following terms apply:

auditable library: complete code and its dependencies of a library is available for free to audit

auditable smart contract: code of the contract and all its internal libraries which are available for audit purposes

contract administrator: entity which is responsible for managing and executing the smart contract

NOTE: In the cases, when the smart contract is shared among multiple participants the governance of the PDL is the owner of the contract.

contract expiration time: time when the governance will call the self-destruct clause to destruct a smart contract

contract owner: entity installed the smart contract

eternal contract type: lack in without internal termination function

governance time: governance clock

Home PDL-Network (HPN): when all the permanent nodes belong to the same PDL network

mainchain: formed at the formation of the consortium and is not dependent on any other chain

off-chain contract type: smart contract not installed on the mainchain

on-chain contract type: smart contract installed on the mainchain

oracles: service that sends data to/from a PDL

NOTE: It should not be confused with the commercial company product name ORACLE® by Sun® Microsystems.

replicated contract type: different smart contract versions active at the same time

sidechain: *sub-chain* of the mainchain

smart contract entry functions: smart contract functions which provide access to a contract from outside world

stakeholders: all the parties benefitted from the smart contract deployment, execution and destruction

smart contract timers: timers that keeps track of the smart contract active/inactive time:

- **long-term timers:** lasts the lifecycle of the smart contract. Contract creation to destruction
- **short-term timers:** duration of an execution of a smart contract. contract initialization until its termination

template contract type: contract stored in ledger which are generalized to be reused by several participants through parametrised executions

termination: Suspend a smart contract:

- **termination:** can be reused with different parameters or can be revised with minor changes
  - **natural termination:** after completing the task
  - **interrupt termination:** during the task
- **destruction:** completed its life cycle - cannot be used anymore.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

A/D	Analogue/Digital
ACL	Access Control List
AI	Artificial Intelligence
API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CPU	Central Processing Unit
DLT	Distributed Ledger Technology
HPN	Home PDL-Network
ID	IDentity
IPR	Intellectual Property Rights
ML	Machine Learning
PDL	Permissioned Distributed Ledger
QoS	Quality of Service
SC	Smart Contract
SSL	Secure Socket Layer
TEE	Trusted Execution Environment
UTC	Universal Time Coordinated

4 Define the Properties of Smart Contracts

4.1 Introduction

Smart Contracts are executable codes which are installed on DLTs (i.e. PDLs for the purpose the present document), therefore their characteristics are dependent on their underlying ledger technology. Some of these characteristics such as immutability and transparency are by-design properties of a PDL and hence common to all PDL-types. Smart contracts inherit these properties from PDLs.

In this clause, such challenges which shall be taken care when designing smart contracts are highlighted.

4.2 Challenges

4.2.1 Inherent Properties

4.2.1.1 Immutability

Smart Contracts are immutable, which means an already registered smart contract cannot be modified or deleted and cannot be tampered with. This way, the integrity of a contract is guaranteed; that is to say, a contractual agreement installed as a smart contract on a PDL becomes ossified, and none of the participants can make any changes retroactively. Immutability produces tamperproof contracts and prevents document frauds. However, immutability comes with a cost of scalability and has two significant problems:

- **An expired contract (or smart contract)** - Even a smart contract that is expired, still lives on the ledger and occupies the storage. For example, if a vendor and an operator are in a contract; the contract may be valid/active for some certain duration and will expire. Such contracts if installed as smart contracts cannot be deleted from the ledger, and cause scalability problem.

- **Erroneous contracts (or smart contracts)** - If a smart contract has bugs or errors, it can make unwanted and unintentional, possibly harmful transactions. It is to be noted here that all the transactions either wanted or unwanted are recorded in a PDL. A bug-free and corrected contract may replace the old contract, but records already stored in the PDL cannot be altered.

4.2.1.2 Transparency

In PDLs, all the ledger nodes keep an identical copy of a ledger; this means they all share the same information. As a result, all the transactions are transparent or known to all the participants of the PDL. Hence, none of them can deny the details of a transaction. In certain cases, or events, when some of the participants of a PDL want private dealings, transparency is not required and may not even defeat the purpose of privacy. For example, a sub-group of participants in a large PDL want to do some business and install a smart contract for the contractual terms and do not want to reveal their contractual details to the rest of the PDL users. In a typical PDL every node will have a copy of this contract but here a private smart contract is required.

A possible solution to this challenge would be private chains or private channels, such as implementation of private channels in Hyperledger Fabric, in where smart contracts can be installed on separate, private channels only visible to the sub-group involved in a contract.

4.2.1.3 Auto-Executable

Smart contracts are triggered by a software condition and can even be executed without human intervention. Auto-executable smart contracts provide an automated method of contracts' execution in which parties can install the contracts as smart contracts which are executed by the code itself. However, this property instigates the following challenges:

- **Uncontrollable executions** - Erroneous code can trigger uncontrollable executions. As an example, unwanted automated payments may cause monetary losses or delivery of incorrect amount of goods due to uncontrollable and out-of-order delivery instruction.

- **Malicious executions** - If malicious parties create backdoors to a smart contract, they can execute smart contracts and it may be difficult to stop such executions without a hard fork to the ledger or installing a revised smart contract that blocks further execution of the malicious smart contract.

4.2.2 Interoperability/Ledger Dependency

Smart contracts have a dynamic nature - they often take input, perform executions and record results to the ledger they are installed on, or may send the execution results to other ledgers. Smart contracts may also take inputs from other ledgers. Following are the scenarios when a smart contract will interact with other ledgers (inter-ledger) and within the ledger it is installed on (intra-ledger):

- **A Smart contract's interaction with other smart contracts in the same ledger (intra-ledger)** - Smart contracts within the same ledger can call each other without any need of harmonisation because they all use the same ledger type. The only consideration here is that if an execution of a smart contract is dependent on another smart contract, they shall be sequential such that an execution is not started until the previous execution is completed and its results are recorded. The reason for that sequence is that the results of the previous executions may later be used as inputs for the next contract in the chain.

- **A Smart contract's interaction with smart contracts in other ledgers (inter-ledger)** - A smart contract may send execution results to another ledger, but the smart contract should have correct access rights to the other ledger. Moreover, both of the ledgers may have different and incompatible data formats which should be addressed. PDL inter-ledger interoperability is discussed in detail in ETSI GR PDL 006 [i.4].

4.2.3 Scalability

This problem is not limited to smart contract and is applied to every aspect of PDL, such as data blocks. Since any data or contract loaded to PDL stays there for the lifetime of the ledger the ledger keeps growing, the ledger will eventually require compute/storage resources that will prevent scale.

For example, in the context of smart contracts, if a consortium of telecom operators runs a ledger to offer service contracts to their customers, this ledger may be running for several years and in those years millions of contracts may be issued. If old and unused contracts are not deleted and removed but can be only deactivated, the ledger will be cluttered with several unused and dormant contracts and ledger resources will be wasted.

4.2.4 Synchronization of Offline Smart Contracts

In a typical PDL, transactions and smart contracts are installed on distributed nodes and these nodes connected to form a ledger to take part in consensus (i.e. approve or reject transactions). In the situations, when some of the nodes go offline possibly due to the reasons such as network connection or duty cycle, there are many scenarios possible, discussed in detail in ETSI GR PDL 010 [i.2], clause 6.2. Two examples are highlighted here:

1) **Independent smart contract** - which may depend on authenticated data from offline nodes (i.e. nodes not connected to the PDL). Such smart contracts may or may not proceed processing depending on same.

2) **Chained smart contracts** - when smart contract execution is dependent on other smart contract execution, then execution will not continue/commence until the required number of nodes are back online.

4.2.5 Ledger Time Synchronization

Like all distributed systems, PDL nodes are distributed across several time zones and do not have solitary clock. This may have several aspects such as local clock of the machine which may or may not be synchronized with atomic clock resulting in inconsistent timestamp. Furthermore, time zone needs to be included to compare with the universal time used for governance timing, including other constraints such as daylight saving.

4.3 Actors in a Smart Contract

See clause 5.2.

4.4 Smart Contract Policy Governance

For role of governance in smart contracts see clause 8 and for details on the general governance role see ETSI GS PDL 012 [i.3].

5 Requirements for Designing a Smart Contract

5.1 Smart Contract Facets

5.1.1 Introduction

Smart contracts are not monotonous, they may take different roles and perform a wide range of operations within and outside the PDL. Following are the roles a smart contract can take.

5.1.2 Foundational Role

Defines the roles, statements, constitution. These types of smart contracts start with the PDL itself and may be the part of the genesis, that is, initialization of the PDL. For example, automated governance can be defined as the functional role.

5.1.3 Functional Role

Smart contracts work as active functions, for example, Access control and intra-circumstances during a PDL.

5.1.4 Business/Operational Role

Mix of both functions - some of the smart contracts have both foundational and functional attributes. For example, monitoring smart contract may initialized with the PDL and performs operations such as access control for its lifetime or lifetime of the PDL.

5.2 Actors

5.2.1 Introduction

All the actors within the PDL network shall be assigned unique identities and access control rights. The governance is responsible to ensure that all the actors are allocated unique access rights, the role of governance is outside the scope of the present document.

The actors related to smart contracts are chosen by the governance and defined as follows.

5.2.2 Lifecycle Management

Lifecycle Management of the PDL is performed by a committee or group of participants (i.e. Governance) chosen by the PDL members by mutual consensus. Typically, management decisions such as access rights and protocols PDL members will adhere to.

Lifecycle Management can be single party or multi-party and the role of Lifecycle Management (i.e. the governance) in smart contract are detailed in clause 8.

5.2.3 Owner

Contract owner is the party who programs and installs the smart contract. In some scenarios, for example, when a smart contract is expected to be shared among several PDL participants, the governance of the PDL can be the owner of the contract.

5.2.4 Stakeholders

All the parties involved in the smart contracts' executions, for example, two contractual partners.

The can different categories of stakeholders:

- Contracting parties - the parties sign the contracts.
- Beneficiaries - the parties affected by the contract/advantage/disadvantaged.

5.2.5 Requirements During Design

5.2.5.1 Lifecycle

Smart contracts are expected to follow the complete lifecycle proposed in clause 4.5 ETSI GR PDL 004 [i.1]. The stepwise approach proposed will facilitate an error-free design of smart contracts. The main advantages of adopting such approach are:

[RLCD 1] *Access Control and Ownerships* - ownership and access control strategies decided during the planning phase will prevent future disputes. This will also facilitate the developers to accurately code the assigned rights while coding the smart contracts. Access Control and Ownership **shall** be defined, discussed, and agreed between the stakeholders and the governance before smart contract coding starts. It is the governance responsibility to ensure this.

[RLCD 2] *Reusability* - smart contracts **shall** be reusable and parametrised for economical storage. During the planning phase, the stakeholders **shall** adopt strategies to design parametrised smart contracts to enable maximum reusability. It the developers' responsibility to ensure a reusable contract.

[RLCD 3] *Minimize human error* - human errors may cause erroneous contracts and may result in a security breach of smart contracts. For example, if a developer mistakenly makes the execution function inaccessible, the contract will never be executed. A smart contract **shall** be tested before the deployment and as specified in clause 9.

NOTE: Human error, such as developer mistakes, may be alleviated through methodical development practices. This occurs during two stages of the smart contract life cycle:

1) the planning phase - by carefully outlining the requirements from the smart contract; and

2) the development and testing phase - by testing the smart contract code against the requirements.

[RLCD 4] *Pre-installation checks* - smart contract **shall** be checked before the final deployment. See clause 9 for details.

[RLCD 5] *Online auditing/monitoring* - smart contracts shall be audited during their execution. See clause 9.6 for details.

5.2.6 Available Technologies

Smart contracts are expected to be widely adopted; hence they should be cautious towards:

[RAT 1] *Programming Languages* - programming language for a smart contract programming is usually ledger dependent but, if possible, widely available, and widely adopted programming languages **shall** be used.

EXAMPLE: In Hyperledger Fabric, developers have choice between several languages (e.g. Golang, JavaScript), in such cases, widely available programming language should be adopted. This will be advantageous to the PDL consortium members in the future as well, for example, it will be easier to recruit developers.

[RAT 2] *Language Libraries* - programming languages often have external libraries, used for different functions such as hashing or digital signing. These external, third-party libraries may include functions which can cause danger to a smart contracts' security. Only governance authorized and verified libraries **shall** be used.

NOTE: If a developer does not do as recommended, it would fail the subsequent audit.

5.2.7 Auditability

5.2.7.1 Auditable Code and Libraries

[RUAL 1] Developers shall use **auditable** libraries for smart contract programming for the purpose of verifiable smart contracts' program/code. Such libraries shall be testable through governance approved testing techniques (e.g. Certification Laboratory using an approved test suite).

[RUAL 2] The Auditable libraries used in smart contract programming and the actual smart contract code **shall** be available for **free use** for auditing purpose.

However, users/developers may or may not pay to use them. The use of open-available and free and the auditability of software libraries will allow inspection and versioning of code in cases of future disputes or malfunctioning of a smart contract.

5.2.7.2 Auditability of Implementation

[RAI 1] All the transactions, communications and dealings of smart contracts, online storage and offline storage are available for audit to interested stakeholders and the governance.

[RAI 2] All the audit available for free-of-cost for audit purpose.

[RAI 3] Data should be stored for length of time as defined by the legal requirements of the application field.

5.2.8 Input to the Smart Contracts

[RINSC 1] Smart contract developers **shall** ensure that a smart contract only accepts input from authorized sources (e.g. authorized APIs).

[RINSC 2] These sources **shall** be approved by and given access rights by the governance functions of the PDL.

The inputs to smart contracts are detailed in clause 7.

5.2.9 Universal Clock

PDLs lack universal clock mechanism due to distributed nature of the nodes.

Smart contracts shall follow:

[RUC 1] Smart contracts shall use Governance defined clock - the time/zone format the PDL network governance.

[RUC 2] Clock of the node may differ from the governance clock and is local to the machine/hardware. In such a case the owner of the node **shall** ensure the synchronization with the governance clock.

[RUC 3] Node shall drive the time from an atomic clock or from another node designated as a source clock (timing source). All nodes shall use the same time specified by the governance. This is to avoid time mismatch between nodes.

[RUC 4] The nodes have the capability to follow and note the PDL time specified by the governance, even if it deviates from the local time (geographical time).

EXAMPLE: Governance may have UTC as its time and all the nodes shall use UTC time as their time.

5.2.10 Terminatable

Eternal contracts can cause problems such as unwanted executions and unauthorized future access. There should be a mechanism to terminate or deactivate smart contracts after a certain date/time. See clauses 6.3.4 to 6.3.6 for details on smart contract termination and destruction.

[RTSC 1] If a developer does not provide a mechanism to deactivate the smart contract, then it shall be known before the deployment of the contract. Consequently, it is advised to have a management action to perform the same.

[RTSC 2] A smart contract **shall** be terminatable.

[RTSC 3] A smart contract shall include a function that can terminate the smart contract. See clause 6.3.4 for details.

[RTSC 4] The owner and governance shall ensure that the parties execute the contract, should also safely terminate it as per specifications in clause 6.3.5.

5.2.11 Security

Security of a smart contract is an important matter because insecure smart contract may allow unauthorized parties to access the data and perform executions. See clause 11 for details.

6 Solutions for Architecture and Functional Modelling

6.1 Introduction

Smart contracts are designed to enable secure executions of the contracts. This clause highlights the architectural and functional requirements; also, solutions for designing a secure smart contract.