***.Disclaimer***

The present document has been produced and approved by the <long ISGname> (<short ISGname>) 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.

ETSI GS PDL-011 v0.0.2 (2021-06)

**Group Specification**

Permissioned Distributed Ledger (PDL);

Specification of Requirements for Smart Contracts’ Architecture and Security

Release #

<

Reference

PDL-11

Keywords

PDL, Policies, Smart Contracts

***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 - NAF 742 C

Association à but non lucratif enregistrée à la

Sous-préfecture de Grasse (06) N° 7803/88

***Important notice***

The present document can be downloaded from:
[http://www.etsi.org/standards-search](http://www.etsi.org/standards-search#Pre-defined Collections)

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:
<https://portal.etsi.org/People/CommiteeSupportStaff.aspx>

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.

The content of the PDF version shall not be modified without the written authorization of ETSI.

The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI yyyy.

All rights reserved.

**DECT**TM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are trademarks of ETSI registered for the benefit of its Members.
**3GPP**TM and **LTE**TM 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.

Contents

Intellectual Property Rights 5

Foreword 5

Modal verbs terminology 5

Executive summary 5

Introduction 5

1 Scope 6

2 References 6

2.1 Normative references 6

2.2 Informative references 6

3 Definition of terms, symbols and abbreviations 6

3.1 Terms 6

3.2 Symbols 7

3.3 Abbreviations 7

4 Define the Properties of Smart Contracts 7

4.1 Introduction 7

4.2 Challenges 7

4.2.1 Inherent Properties 7

4.2.1.1 Immutability 7

4.2.1.2 Transparency 7

4.2.1.3 Auto-Executable 8

4.2.2 Security 8

4.2.3 Interoperability/Ledger dependency 8

4.2.4 Scalability 8

4.2.5 Synchronisation of Offline smart contracts 9

4.2.6 Ledger Time Synchronisation 9

4.2.7 Congestion/Execution Latency 9

5 Requirements for Designing a smart contract 9

5.1.1 Requirements during designing 9

5.1.2 Lifecycle (can be address as a part of requirements) 9

5.1.3 Available Technologies 10

5.1.4 Input to the Smart Contracts 10

5.1.5 Universal Clock 11

5.1.6 Terminatable 11

6 Solutions for Architecture and Functional Modelling 11

6.1 Introduction: 11

6.2 Architectural and Functional Requirements 11

6.2.1 Template Contracts: 11

6.2.2 Internal Termination: 11

6.2.3 Self-Destruct: 12

6.2.4 Internal Timer: 12

6.2.5 Secure Access control mechanisms: 12

6.2.6 Entry Functions: 12

6.2.7 Control Signalling 12

6.2.8 Solutions 13

6.2.9 Standard Fields 13

6.2.10 Mandatory Fields: 13

6.2.11 Optional Fields: 13

6.3 Architecture 13

6.3.1 Template Contract: 13

6.4 Architecture and Functional Descriptions 14

6.5 Data Inputs 15

6.5.1 Internal Smart Contracts 15

6.5.2 Oracles (should be externals –we should use another term) 15

6.5.3 Other data sources 15

7 Governance (may be PDL-focused rather than SC?) 15

7.1 Introduction 15

7.2 Single-Party Governance 15

7.3 Multi-party governance 15

8 Testing Smart Contracts 15

8.1 Generalised testing targets 15

8.2 Offline Testing 16

8.2.1 Sandbox testing 16

8.2.2 Testbeds 16

8.3 Online Monitoring 16

9 Updating a smart contract 16

9.1 Introduction 16

9.2 Versioning 16

9.3 Securely Inactivating old contract 16

9.4 Technology specifications 16

10 Threats and Security 16

10.1 Introduction: 16

10.2 Threats 16

10.2.1 Internal Threats 16

10.2.2 External Threats 17

10.2.3 Accidental damages 17

10.2.4 Malicious attacks 17

10.3 Mitigation Strategies 17

Annex A (normative or informative): Title of annex 18

Annex B (normative or informative): Title of annex 19

B.1 First clause of the annex 19

B.1.1 First subdivided clause of the annex 19

Annex (informative): Bibliography 20

Annex (informative): Change History 21

History 22

# Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is 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](https://ipr.etsi.org/)).

Pursuant to the ETSI IPR Policy, no investigation, 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.

# Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (<short ISGname>).

# 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](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx) (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

This Work Item 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

This work item extends the discussion of challenges and requirements for the successful adoption of smart contracts. This 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, this document also oversees smart contracts' security aspects and explains internal and external threats to a smart contract and presents possible mitigation techniques for them.

# Scope

This work item establishes the architectural and functional specifications of smart contracts. Additionally, highlight the potential threats and specify the solutions to mitigate them. Requirements on the use of technology for smart contracts, governance, purpose, motivation and security.

# References

## 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](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.

[1] <Standard Organization acronym> <document number>: "<Title>".

[2] <Standard Organization acronym> <document number>: "<Title>".

## 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

“DGR/PDL-004\_smartcontract” v1.1.1 08th Feburary 2021

available from: https://www.etsi.org/deliver/etsi\_gr/PDL/001\_099/004/01.01.01\_60/gr\_PDL004v010101p.pdf

[i.2] etc.

# Definition of terms, symbols and abbreviations

## Terms

For the purposes of the present document, the [following] terms [given in ... and the following] apply:

**Eternal Contracts:** Those smart contracts lack in without internal termination function.

**Replicated Contracts:**

**Template Contracts:** Contract stored in ledger which are generalised to be reused by several participants through parametrised executions

**Entry Functions:** Provides access to a contract from outside world.

**Contract Owner:** The entity installed 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

**Governance Time**: Governance clock

Auditable Libraries: Programming language libraries which are available for free of charge for audit.

## Symbols

For the purposes of the present document, the [following] symbols [given in ... and the following] apply:

## Abbreviations

For the purposes of the present document, the [following] abbreviations [given in ... and the following] apply:

# Define the Properties of Smart Contracts

## Introduction

Smart Contracts are executable codes which are installed on DLTs (i.e., PDLs for the purpose this 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 must be taken care when designing smart contracts are highlighted.

## Challenges

## Inherent Properties

## Immutability

Smart Contracts are immutable, which means an already installed 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.

## 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

A possible 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.

## 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* e*xecutions* – 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.

## Security(should we remove it, we have a dedicated chapter)

## Denial of Service (DoS) Attack

Denial of Service attack is, when a malicious user sends multiple and unwanted requests to the PDL simoultaneously or at a very slim interval that the PDL can cope with. This is type of attacks are very unlikely in PDLs, because of the permissioned nature of the PDLs. Moreover, all the parties in the PDL are known to the governance and are allowed with strict access control mechanisms.

## Reentrancy Attack

## Numerical/Integer Overflow/Underflow Attack

Numerical Overflow attack is similar to the classic Integer Overflow problem in computer programming. For example, in programming languages such as Solidity, the maximum size of an integer value is 256 bytes. Beyond this limit, the system will throw an Integer Overflow Exception. In such a case, if an attacker intentionally sends a massive input is given to a smart contract. Within the contract, if some operation makes the input size and produces another output larger than 256 bytes, it can cause integer overflow. In such a case, it may result in an unexpected large size of transfer to the attacker.

Strategies to minimise the human error

* Parity multisig wallet attack
* Integer overflow/underflow attack

## 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 harmonization 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 must 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 clause xx of PDL-006.

## 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 run 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.

## Synchronisation of Offline smart contracts

In a typical PDL, transactions and smart contracts are installed on distributed nodes and these nodes have to be connected to the 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 PDL 010, clause xx [i.x]. more details? Two examples are highlighted here:

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

## Ledger Time Synchronization

Like all distributed systems, PDL nodes are distributed across several time zones and don’t have solitary clock. This may have several aspects such as local clock of the machine which may or may not be synchronised 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.

## Congestion/Execution Latency

* Smart contract execution times can cause congestion at the ledger – governance defined time to handle this?
* Use concrete word for ledger/Smart contact, because dependent on the ledger technology.
* Issue must be clear in the document on the ledger technology (IPR issue).
* Should consider whilst adopting smart contracts’ applications – the ISG doesn’t advocate any particular ledger technology type
* (more details)

# Requirements for Designing a smart contract

## Requirements during designing

## Lifecycle (can be address as a part of requirements)

Smart contracts are expected to follow the complete life cycle proposed in clause 4.5 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:

* *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.
* *Reusability* – smart contracts are required to be reusable and parametrised for economical storage. During the planning phase, the stakeholders can adopt strategies to design parametrised smart contracts to enable maximum reusability.
* *Minimise 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.

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.

* *Comprehensive Testing – The two phases that enable comprehensive testing are 1) Planning phase – the* required tests should be outlined and 2) Design and Testing phase – A smart contract shall be tested thoroughly, and the developers should ensure that all the required planned tests have been successfully completed.
* *Pre-installation checks –* duringsmart contracts’ coding and testing phase, a smart contract shall be checked before the final deployment. This will reduce risks oferroneouscontracts.
* *Online auditing/monitoring –* smart contracts should be audited during their execution, but since smart contracts are immutable, they cannot be changed, even if they are not performing as planned. The smart contract, however, can be terminated through executive access rights.
* 

Figure 5‑1: Life cycle of smart contract – Activation and Self-Destruct

## Available Technologies

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

*Programming Languages –* programming language for a smart contract programming is usually ledger dependent but if possible, widely available and widely adopted programming languages should be used. For 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.

*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 contract security. It is recommended that developers 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).

NOTE: If a developer doesn’t do as recommended, would fail the subsequent audit.

## Usage of Auditable Libraries

Auditable libraries 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.

To be removed:

* Libraries will enable audit the code
* They will be available for the audit purpose but people may/may not pay to use them.
* Can be restricted auditing,
* Can help with versioning of code

## Input to the Smart Contracts

Smart contract developers shall ensure that a smart contract only accepts input from authorised sources (e.g., authorised APIs). These sources shall be approved by and given access rights by the governance functions of the PDL. The possibilities of inputs and considerations are:

***Possible Inputs:***

* Internal input – internal contracts and APIs, that is to say input originated from the same PDL can access a smart contract with appropriate access rights, defined by the governance and the owner of the contract. Access and number of concurrent requests to a smart contract is defined through a PDL governance, that maintains the record of all access rights and ensures the prevention of congestion. For example, if the governance of a PDL is unaware of the concurrent number of requests for a smart contract, they may allow additional external inputs which can cause congestion at the ledger.
* Oracles (External Input) – smart contracts shall not access external sources directly but may use an **intermediate API** that provides an extra access control layer between the smart contract (hence the PDL) and the other sources. Auditing of the incoming data? Who to ask and to whom?
* The incoming data from oracles shall be checked for compatibility such as fields format, data types etc. The details of inputs are further discussed in clause XX.

***Considerations:***

* Sequencing and Synchronising inter-ledger and intra-ledger executions – (e.g., Output from one ledger is input to another ledger). Some smart contracts are dependent on operations/inputs from other smart contracts, which may be internal or external. Smart contract execution time depends on the underlying PDL type, and a smart contract dependent on such inputs shall allow sufficient time to complete the execution or wait for a signal from other ledger before further execution commences.
* Quality of the data (e.g. syntax, semantics, context )

## Universal Clock

PDLs lack universal clock mechanism due to distributed nature of the nodes. In this case, a universally acceptable clock is required to ensure that

* Governance defined clock (more details needed - ISO time standard – universal time zone – PDLs shall comply in the future).
* Relative to location of the node.
* Shall be calibrated to an atomic clock. Governance can specify the granularity/level of accuracy.

## Terminatable

Eternal contracts can cause problems such as unwanted executions and unauthorised future access. Hence there should be a mechanism to terminate or deactivate smart contracts after a certain date/time.

NOTE: if a developer doesn’t 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.

# Solutions for Architecture and Functional Modelling

## 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.

## Architectural and Functional Requirements

## Template Contracts

Smart contracts are immutable and if there is an error (e.g., developers’ mistake), they may need to be obsoleted by a newer improved smart contract. In order to avoid that, it is recommended to use template contracts that have been tested and debugged to reduce the chance of such errors. Template contracts are designed with more consideration and secured with several security checks. Templates may require specification of the throughput/bandwidth of the contract and depend on the governance, application of the contract and the stakeholders involved.

## Internal Termination

As discussed in clause 5.1.7, Eternal smart contracts can cause unwanted executions in future. Hence, they should be terminated with an internal termination signal/command. Every smart contract should have a termination function that disables further executions of the smart contract after a certain period or upon notice. That is to say, if a contract is generalised enough to be shared among several users, such contract should have means of terminating/deactivating itself after an end-date or upon a specific condition.

Note that, that a deactivated smart contract may be reactivated for future contract executions with different parameters upon signal or specific conditions.



Figure ‑: Smart Contract Internal Termination

## Safe Termination:

The smart contract can be assigned very sensitive tasks, so it is important to safely terminate them to avoid future attacks. In such cases governance and owners of the contract, shall ensure the safe termination. That is:

* All the functions shall be inactivated before the self-destruct
* Ensure that none of the smart contract function is callable.
* All the access rights of the smart contracts are revoked.
* Data is archived for future reference such that data integrity should be maintained.

## Self-Destruct

Self-destructed contracts differ from internally terminated contracts, by them not being able to reactivate. This means, once destroyed the contract cannot be reinstated. In a typical PDL, a smart contract cannot be initiated on its own, therefore, self-destruct must be initiated. The governance of the PDL can initiate the self-destruct through a control instruction (clause 6.2.7).

NOTE 1: It is possible that some developers/owners install a new version of smart contract after a self-destruct, in such cases, they shall follow the old versioning sequence.



Figure ‑: Smart Contract Self-Destruct. Note that, Self-Destruct instruction shall be sent explicitly and externally

## Internal Timer(I think it should be removed, ask in meeting and then remove it)

Smart contracts must have an internal timer, which shall starts ticking on condition (typically that would be the first deployment of the contract). This internal timer facilitates the successful deactivation/destruction of the smart contract based on self-destruct or internal termination clauses herewith described. Relation between node sync and internal timer? Transferable replication. Get the details on clock sync in terms of nodes, for example some nodes are not syncd what will happen?

## Secure Access Control Mechanisms

Smart contracts should be accessible by the owner(s) of the contract or any other party authorised/delegated by the governance. A stringent access control mechanism should be implemented to enforce the same.

For example, in some cases, the owner of the contract or governance may temporarily use **delegates** to assign rights of a contract. If the delegation is changed, a delegate will have most of the rights as the owner but not all. More specifically, a delegate may not be allowed to further delegate the smart contract rights to another party. Such delegation rights will stay exclusive to the owner and the governance of the PDL. However, an owner may allow the further delegation to the delegates with discretion in some situations.

It is possible that, some of the delegates further delegates the access rights without authorization. To handle this, the device authentication should be implemented. That is to say, the access keys assigned to the delegate shall also check for the device identification. If a different device is identified than the authorized one than the access of the delegate shall be blocked, and warning should be issued.

NOTE: Only some of the rights can be delegated to other members of the PDL. For example, governance functionality cannot be delegated.

NOTE: The delegation rights should be the function of the Governance Layer, for example, the governance of some PDLs may allow delegation of all functionalities and the governance of some PDLs may prefer to restrict delegation of some functionalities only.

Access delegation should be for a limited time and will be revoked when such time elapses. The delegate identification can be done through node identifier which is assigned by the governance to the nodes of the PDL.

Ismael: Agents – delegation for secure access, they may get incentivised to do so.

## Entry Functions (Required Functions)

In a typical case, contract users shall be able to access only a certain subset of functions. Interaction between smart contracts’ functions should be limited and controlled to prevent unauthorised access to other users’ data. For example, if a contract function initiates a payment to a client after checking the user credentials through check\_access and then invokes payment through a payment function, this function should only issue the payment and shall not invoke/initiate any other function of the contract, for example, historical data of payments.



Figure 6‑3: Entry Functions shall not create back doors to other functions of the contract

## Control Instructions

Like every software code, smart contract shall need instruction during a contract execution. Though these are function which are executed in special circumstances, other than usual smart contract operations and are only for the purpose of control signalling. For example, sending an interrupt instruction. Some of the examples to send control instructions are as follows:

* *Blocking/Unblocking Instruction* – if a smart contract is not performing as planned, may be reasons such as error in a contract, such contract should be stopped or blocked by exclusive instructions. Depending on the requirements of the contract, such instructions can be of several types such as, invoke/execute the termination clause
* *Function updates - depends on the PDL type, for example, some functions are important see Table 1*
* *Reconfiguration Instruction* – reconfiguring the data of the smart contract. For example, extend the end date of the auction contract. *(may be part of minor function updates remove it?)*
* *Self-destruct Instruction – As discussed in clause 6.2.3*



Figure 6‑4: Instruction Flow Between the Layers (harmonise it with PDL 003)

Table : Function Instructions

|  |  |  |  |
| --- | --- | --- | --- |
| **Functions** | **Governance** | **Owner** | **Delegate** |
| Change of owner | yes | No (see NOTE 1) | No |
| Interrupt | Yes | Yes | Yes |
| Terminate | Yes | Yes | Yes |
| Access duration | Lifetime of the contract | Limited (see NOTE 2) | Limited (see NOTE 3) |
| Delegation | Yes | Yes | See clause 6.2.5 |
|  |  |  |  |
|  |  |  |  |
| NOTE 1: The owner may ask the governance of the PDL for the change of the owner.NOTE 2: The owner of the contract has governance defined access durationNOTE 3: The governance will delegate limited time duration NOTE 4: Delegation should be the functionality of the Governance layer |

Do we need governance layer to manage the smart contracts? Governance layer may only be applicable to access rights only. Find some examples of present architecture of governance.

## Archiving the data

Smart contracts are only execution code, the data generated by them is recorded in the ledger. In some cases, the customers may wish to terminate the contract completely, such as self-destruct function in Ethereum, which completely removes all the states from the ledger. Note that, such contracts can still be reinstated.

 However, in the following cases the owners shall ensure to *Safely Terminate* (clause 6.2.3) the contract.

* Following are the example cases where the safe termination may be important.

|  |  |
| --- | --- |
| * **Contract Type**
 | * **Importance of data archiving (e.g., the sensitive data this function may have)**
 |
| * Financial Contracts
 | * payment functions
 |
| * Identity Contracts
 | * Public keys, certificates
 |
| * Auction Contracts
 | * Bid values, minimum bid value
 |
| * Service Level Agreement Contracts
 | * Contract between vendor and operators
 |

In some cases, however, the owner may wish to keep the local record of the smart contract and its executions. Note that, in such cases integrity of the data is very important. Other participants may not trust the locally stored data of the smart contract. Following are some of the storage methods owners may wish to use to store their data:

* Back up the executions and data in a cold storage with timestamped xxx
* Create a side-chain, and copy the executions there,
* Hash the smart contract and the respective data and store it in a local storage.

## Stale data

If a smart contract is self-destructed/ deactivated the old data will be there? How to handle it?

## Updates

Smart contracts will be updated, as required. These updates shall maintain the standard fields flow such as continuous and correct versioning. The following ways a smart contract can be updated:

* *Creation after self-destruct:* If a smart contract is self-destructed, and a new version of the smart contract is created, it will also be categorized as an update to the contract. In such cases, it should maintain the versioning sequence of the prior smart contract. That is, for example, if the self-destructed smart contract was v0.0.1, then the newly created contract shall be v0.0.2.
* *Update* *with new parameters:* If a smart contract is terminated (clause 6.2.2) and executed again with new parameters. In such cases, the version number shall remain same but *Execution ID* (clause 6.2.13) changes.

## Standard Fields

### Mandatory Fields:

(PDL-003)

Fixed/Permenant:

* *Contract ID* – an internal identity of smart contract. This identity is allocated by the governance of the PDL. E.g., LedgerID: ContractID: OwnerID
* *Ledger ID* – Every ledger shall be identified by a unique ID, all the regulated ledgers should be assigned an ID by the relevant authorities(who?) e.g., CompanyID: ReigonID
* *Owner ID* – Typically a smart contract should be owned by the governance and borrowed/leased to users of the PDL. However, it is possible that some users wish to install customised contracts for specific purposes. In both the cases, the owner ID should be the mandatory section of a smart contract.
* *Start time (Governance-defined clock)* – the start time of the contract, this is time when the contract is deployed in the ledger and is different from the execution/invocation time of the ledger. Default time when the smart contract is deployed. May consider defaulting it. Life cycle diagram
* *End time (All times will represent to governance-defined clock)* – the end time of the contract, the self-destruct clause will execute at this time and the contract will be terminated automatically. Users of the contract shall ensure that all the sub-contract execution time is before the end-time of the contract. Depends on the scenario, it may be changeable.
* *Version No.* – Version Number of the smart contract, if a smart contract is re-deployed after self-destruct, same versioning sequence shall be followed.

Parameterised:

* *Execution Start Time –* start time of a particular smart contract execution
* *Execution End Time* – end time of a particular smart contract execution. Must be before the contract end time.
* *Execution ID* – identity of execution by a user of the PDL or external entity.
* *Executing party ID –* Identity of the participant executing the smart contract. This is different from the public key and is the permanent identity assigned by the governance of the PDL. In a typical PDL, the transaction ID or public key of the executing participant is recorded in the ledger at the time of the contract execution, but it should be the part of the contract as well. Anonymisation should be resolvable because we support accountability in PDLs. Pseudo-anonymised by the PDL governance. For GS some ways to specify the ways to generate these IDs.

### Optional Fields:

Optional fields of a smart contract depend on several reasons such as purpose, usage and timing. Nevertheless, we outline some of the examples for the reference:

Description of every block in a normative way (optional).

## Architecture

Importing data

How the data is tracked?

Life-time of the contract?

### Template Contract:

### Initialisation:

The contract is initialised with the initialisation clause. The initialisation clause may include initialization of variables or verifying the login credentials.

### Logic:

The main working of the smart contract. It may include several classes and functions.

### Termination:

As explained in clause xxx, the smart contract must be terminated to avoid future accidental executions. The developers of the contract terminate the smart contract, with some functions in this clause

### Timer:

The internal timer of a smart contract. This is infact a function inside a template contract

* The timer function must be activated by an external call
* Some APIs for Ethereum are available but we want to do this





## Architecture and Functional Descriptions

* Sub diagrams and details of every block – should be very detailed.
* Initialisation
* Logic
* Termination
* API (Optional)
* Timer
* Self-Execution
* Oracles (careful not to talk about the Sun Systems ORACLE)

## Data Inputs

## Internal Smart Contracts

* Requirements the data entered by other smart contracts
* Which fields of the smart contract will be accessible by other smart contracts?
* Who will govern the access?
* Access control mechanisms? Life-long or task based?

## Oracles (should be externals –we should use another term)

* Simple specification of oracle itself, and the clarification that it is not the ORACLE- Sun microsystems.
* Find another term oracles- may be users of the same PDL or APIs if they are entering the record
* How they will access the ledger
* How the access will be granted and for how long? – depends on the governance
* Access control mechanisms – who will assign the access? Is it task limited? If a task is completed the keys should be revoked? Or it is lifelong?

## Other data sources

* Decentralised Applications (like a marketplace)
* Other ledgers – will access the SC with an API
* Direct user interactions – should not be allowed – only through an API or DApp

# Governance (may be PDL-focused rather than SC?)

## Introduction

Define that governance is a PDL problem and here we discuss only in the context of SC and how it will affect the management decisions of SC such as access control and ownership.

## Single-Party Governance

* Access control mechanism
* Decisions? Who will own the SC? And how the access be granted?
* Default access duration?
* Clock – which clock to follow? Relative to the node? Or the governance clock.
* Explain the possible wrong doing of a smart contract.

## Multi-party governance

* Clock management – in PDLs clock is a problem because the nodes are distributed – in multi-party governance which clock to follow
* Access control – who will own and what?
* Consensus to assign rights – voting?

# Testing Smart Contracts

## Generalised testing targets

* Check for lifecycle – the reference is the life cycle – so here we do the testing with respect to the life cycle we did in GR
* Audit

## Offline Testing

## Sandbox testing

* Run a local copy of the ledger

Requirements and specifications for sandbox testing

## Testbeds

* Run a distributed actual ledger locally as a test-bed

Requirements and specifications for smart contract test beds

## Online Monitoring

* Reporting online bugs
* Specify a protected function in a smart contract which is open to the owner only and can be used to destroy a smart contract if it is buggy or has some unwanted behavior
* Suspension and upgradation of smart contract

# Updating a smart contract

## Introduction

## Versioning

* Must follow the correct, governance-defined versioning whilst updating a smart contract (avoid random versioning)

## Securely Inactivating old contract

* Backing up the old version
* Check for technology updates - any updates are required in order for the contract to be visible in the future.
* New version should follow pattern of the old version – interoperable with old versions. Should be accessible with minimal changes to the code/technology.

## Technology specifications

* Must follow upto date technology requirements
* If a vulnerability found should be updated

# Threats and Security

## Introduction:

## Threats

## Internal Threats

Threats due to the home PDL and SCs

* Transactions ordering
* Malicious users in the PDL, very unlikely because of the strict access control mechanisms of PDL
* Open functions of a smart contract may give peek to other user in the ledger – likely if not caught, these kind of problems are particularly important to address in the context of channeling
* Faulty incorrect invocation – checked thoroughy whilst programming it (e.g., invalid Contract ID/Ledger ID)
* Malicious Delegate

## External Threats

* Faulty data entered in a smart contract?
* Too many transactions – may not be able to process by a smart contract?/ congestion at the ledger
* Authorised access – the keys should be revoked without ANY delay.

## Accidental damages

* System should be able handle accidental attacks and minimise such problems

## Malicious attacks

* System should be self-protecting
* malicious attacks should be penalised.

## Mitigation Strategies

* Dependent on the local laws – should follow the governance.
* Checking for invocations only API/DApp layer – we need to make sure they don’t enter any invalid data
* Malicious acts such as wrong data is very difficult to pickup. In that case we have online monitoring, the owner can step in and revoke the contract
* Detect them after the damage – black list/ block the node. Penalties/compensation can be imposed on the malicious nodes.
* Step wise approach – first offence, second offence and so on. At the end node can have lifetime ban or high penalities.

## Bandwidth and throughput trade-off (it is not requirement)

Secondly, Smart contracts cannot be deleted from the ledger, therefore installing replicated contracts may overcrowd the ledger. Hence, when possible, generalised templates of smart contracts are installed on the ledger. These template contracts may be owned by a group of PDL participants or the governance of the PDL. Templates are valuable and should be used – hence it is important to standardise them through governance and standardisation bodies. They are not mandatory but will make the process easier and uniform. Here, it is a trade-off, installing one contract may effect simultaneous executions of the contract hence, the throughput, that is to say, if several parties want to access a smart contract concurrently, they have to wait until one execution is completed. In such cases, when concurrent execution is required, it may be feasible to install several smart contracts to increase throughput of the contract/ledger.

Annex A (normative or informative):
Title of annex

Annex B (normative or informative):
Title of annex

# B.1 First clause of the annex

## B.1.1 First subdivided clause of the annex

Annex (informative):
Bibliography

Annex (informative):
Change History

| Date | Version | Information about changes |
| --- | --- | --- |
| 03/2021 | 0.0.1 | Initial draft with table of contents |
|  |  |  |
|  |  |  |
|  |  |  |

# History

|  |
| --- |
| **Document history** |
| <Version> | <Date> | <Milestone> |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

*Latest changes made on 2019-09-10*