ETSI GS PDL 011 V1.1.1 (2021-12)

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 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
DGS/PDL-0011_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

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

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.

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

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.
No recommendation as to products and services or vendors is made or should be implied.
No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.
In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

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 2021.
All rights reserved.

Contents

Intellectual Property Rights 6
Foreword 6
Modal verbs terminology 6
Executive summary 6
Introduction 7
1 Scope 8
2 References 8
2.1 Normative references 8
2.2 Informative references 8
3 Definition of terms, symbols and abbreviations 9
3.1 Terms 9
3.2 Symbols 10
3.3 Abbreviations 10
4 Define the Properties of Smart Contracts 10
4.1 Introduction 10
4.2 Challenges 10
4.2.1 Inherent Properties 10
4.2.1.1 Immutability 10
4.2.1.2 Transparency 11
4.2.1.3 Auto-Executable 11
4.2.2 Interoperability/Ledger Dependency 11
4.2.3 Scalability 11
4.2.4 Synchronization of Offline Smart Contracts 12
4.2.5 Ledger Time Synchronization 12
4.3 Actors in a Smart Contract 12
4.4 Smart Contract Policy Governance 12
5 Requirements for Designing a Smart Contract 12
5.1 Smart Contract Facets 12
5.1.1 Introduction 12
5.1.2 Foundational Role 12
5.1.3 Functional Role 12
5.1.4 Business/Operational Role 13
5.2 Actors 13
5.2.1 Introduction 13
5.2.2 Lifecycle Management 13
5.2.3 Owner 13
5.2.4 Stakeholders 13
5.2.5 Requirements During Design 13
5.2.5.1 Lifecycle 13
5.2.6 Available Technologies 14
5.2.7 Usage of Auditable Libraries 14
5.2.8 Input to the Smart Contracts 14
5.2.9 Universal Clock 15
5.2.10 Terminatable 15
5.2.11 Security 15
6 Solutions for Architecture and Functional Modelling 15
6.1 Introduction 15
6.2 Smart Contract Offloading 15
6.2.1 Introduction 15
6.2.2 Requirements for the External Storages 16
6.2.3 Requirements for External Smart Contracts 17
6.2.4 Home-PDL Managed Storage 17

6.2.4.1 Non-PDL External Storage 17
6.2.4.2 HPN Managed Sidechain 18
6.2.5 Non-HPN Sources Managed Storages 19
6.2.5.1 Introduction 19
6.2.5.2 Non-HPN Mainchain 19
6.2.5.3 Non-HPN Sidechain 19
6.2.5.4 Third-Party Managed External Storage 20
6.2.6 Modularized Smart Contracts 20
6.2.6.1 Introduction 20
6.2.6.2 Sidechain Offloading 21
6.2.6.3 HPN-Managed Datacentre 22
6.2.6.4 Non-HPN PDL Smart Contract Split 23
6.2.6.4.1 Non-HPN Mainchain 23
6.2.6.4.2 Non-HPN Sidechain 23
6.2.6.5 External and Shared Storages Among PDLs 24
6.2.6.6 Time-Dependent Smart Contracts 24
6.2.6.7 Smart Contract Time/Timers 24
6.2.7 Data Integrity Sanity Checking 25
6.3 Architectural and Functional Requirements 25
6.3.1 Lifecycle of a Smart Contract 25
6.3.2 Template Contracts 26
6.3.3 Time-Limited 26
6.3.4 Internal Termination 26
6.3.5 Safe Termination 27
6.3.6 Destruction 27
6.3.7 Secure Access Control Mechanisms 28
6.3.7.1 Introduction 28
6.3.7.2 Access - Control Requirements for delegates 29
6.3.8 Control Instructions 29
6.3.9 Archiving the data 30
6.3.10 Stale data 31
6.3.11 Updates 31
6.3.12 Smart Contract Fields 31
6.3.13 Mandatory Fields 31
6.3.14 Fixed/Permanent 31
6.3.15 Parameterized 32
6.3.16 Optional Fields 32
6.4 Architecture and Functional Descriptions 33
6.4.1 Required Functions 33
6.4.1.1 Initialization 33
6.4.1.2 Access-Control 34
6.4.1.3 Logic 34
6.4.1.4 Entry Functions 34
6.4.1.5 Termination Function 35
6.4.2 Example Architecture of a Smart Contract 36
6.4.2.1 Smart Contract with External Input 36
6.4.2.2 Smart Contract with another smart contract 37

7 Data Inputs and Outputs 37
7.1 Introduction 37
7.2 Generalized Input/Output Requirements 37
7.3 Internal Data Inputs 38
7.3.1 Introduction 38
7.3.1.1 Internal Inputs Options 38
7.3.1.2 HPN Participants 38
7.3.1.3 HPN Smart Contracts 38
7.3.2 Requirements for Internal Outputs 39
7.4 External Data Inputs 39
7.4.1 Non-HPN 39
7.4.1.1 Introduction 39
7.4.1.2 Faster Approach 39
7.4.1.3 Secured Approach 40

7.5 Oracles....40
7.5.1 Introduction....40
7.5.2 Requirements for Oracles....41
7.5.3 Oracles' Access Rights....41
7.5.4 Oracles as Internal Service....41
7.5.5 Oracle from External Sources....42
7.5.6 Criteria for Oracles Services Approval....42
7.5.7 Offline Oracles....43

8 Governance Role in Smart Contracts....43
8.1 Introduction....43
8.2 Governance Role Delegated to Policy of Smart Contract....43
8.3 Updating a smart contract....44
8.4 Operational Decisions....44
8.5 Termination of contract....44
8.6 General Compliance Strategies for Smart Contracts....45

9 Testing Smart Contracts....45
9.1 Introduction....45
9.2 Testing Strategies....46
9.3 Generalized Testing Targets....46
9.4 Testing Checklist....46
9.5 Offline Testing....47
9.5.1 Introduction....47
9.5.2 Sandbox Testing....47
9.5.3 Testbeds....48
9.6 Online Monitoring....48
9.6.1 Introduction....48
9.6.2 Time-Limited Test....48
9.6.3 Monitoring....48
9.6.4 Online Reports....48
9.6.5 Decisions Based on the Reports....49

10 Updating a Smart Contract....49
10.1 Introduction....49
10.2 Update Situations....49
10.3 Strategies of Updating....49
10.3.1 Old Version....49
10.3.2 Technological Upgrades....50
10.3.3 Upgrading Through Versioning....50
10.3.4 Updating Steps....50
10.3.5 Checklist Before Redeployment....50
10.3.6 Securely Inactivating Old Contract....50

11 Threats and Security....51
11.1 Introduction....51
11.2 Threats....51
11.2.1 Smart Contract Programming Errors....51
11.2.2 Internal Threats....51
11.2.2.1 Transactions Ordering....51
11.2.2.2 Malicious/Accidental Executions....51
11.2.2.3 Reporting Wrong Parameters....52
11.2.3 External Threats....52
11.2.3.1 Introduction....52
11.2.3.2 Malicious Oracles....53
11.2.3.3 Accidental Damages....53
11.2.3.4 Malicious Attacks....53
11.2.3.5 Denial of Service Attack....53
11.2.3.6 Re-entrancy Attack....53
11.2.3.7 Numerical/Integer Overflow attack....53

History....54

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, 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.

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: "PDL Reference Architecture Framework".

NOTE: Available at https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=63501.

[i.4] ETSI GR PDL 006: "Inter-ledger Interoperability".

NOTE: Available at https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=59251.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

auditable libraries: programming language libraries which are available for free of charge for audit

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

contract administrator: entity that is responsible for manage and execute 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 installed not 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

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

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

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 Ledgers
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 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 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 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 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.

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 document 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 be 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 affect 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, would fail the subsequent audit.

5.2.7 Usage of Auditable 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 **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.

5.2.8 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 synchronize 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 noting 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 - 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.

6.2 Smart Contract Offloading

6.2.1 Introduction

In a simplest model, smart contracts are stored in the mainchain as shown in Figure 6-1.

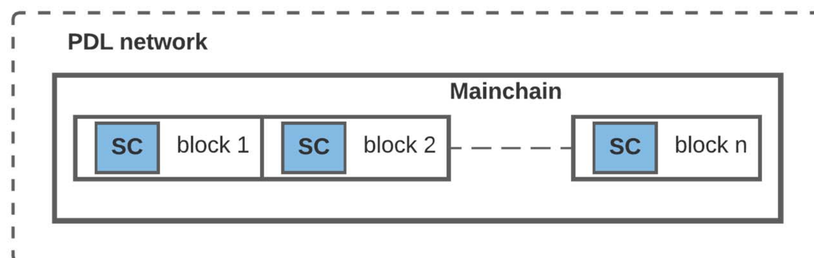

Figure 6-1: Smart contract in a simplest scenario: Smart contracts are installed on the mainchain

However, it is sometimes infeasible and even unnecessary to install the smart contract on the mainchain. One way to implement that will be to off-load all or some of the functionalities of the smart contract to an external storage (e.g. secure data storage).

[COFF 1] ***Management Overheads -*** may require additional management overhead (e.g. storage auditing).

Several combinations of external storages are possible, some are listed here:

Managed by HPN:

- Sidechain.
- Non-PDL storage.

Managed by third party:

- Foreign PDL mainchain.
- Sidechain.
- Non-PDL storage.

6.2.2 Requirements for the External Storages

Regardless of the type of external storage listed in clause 6.2.1 and otherwise, the external storages serving smart contracts shall have following ***requirements:***

[RES 1] The external storage is secured, that is, smart contract execution is safe from all the internal and external, both malicious and benign attacks.

[RES 2] The storage may be shared by several other smart contracts and software, in such case, the smart contract shall work in an isolated fashion.

[RES 3] The external storage shall be tamper resistant as the PDL.

[RES 4] The governance shall list the requirements of storage and all the stakeholders, and the owner of the contract shall approve it.

[RES 5] The list of external storage requirements **shall** include:

a) Hardware requirements, notably, required processor and memory.

b) Software requirements.

c) Security Protocols.

d) The type of smart contracts is expected to be executed on this storage.

e) The duration for which the external storage will be used for the PDL.

[RES 6] It is the storage owner's responsibility to ensure that the storage follows the governance specifications in terms of:

a) Hardware requirements, notably, required processor and memory.

b) Software requirements.

c) Security protocols.

[RES 7] The governance of the PDL, whose contracts are stored in the external storage shall carryout periodic security and standards audit to ensure that correct standards are followed.

[RES 8] The results produced by external execution should be provided to the PDL in a timely fashion.

[OAS 1] The list of external storage requirements **may** include:

a) The number of smart contracts will be running on the external storage.

[OAS 2] Owner/governance of the contract may install the smart contracts inside a TEE or use other secure storage mechanisms.

6.2.3 Requirements for External Smart Contracts

Following are the generalized ***requirements*** when a smart contract is executed on external storage:

[RESC 1] Execution reports shall be sent to the governance periodically for audit purposes.

[RESC 2] Owner/Administrator of the storage shall ensure the security of contracts and their executions.

[RESC 3] Owner/Administrator shall ensure that governance-defined security protocols are followed.

[RESC 4] The smart contract in an external storage shall access the PDL, as per the access control mechanism defined in the clause 6.3.7.

[RESC 5] The governance, owner and all stakeholders shall give an approval for smart contract offloading, in a verifiable manner (e.g. digitally signed document).

[RESC 6] Coordination and synchronization - All the execution blocks shall perform in a synchronized fashion. Execution in external storages may be faster than PDL executions, in such cases developers of the PDL shall ensure synchronization of both the execution blocks.

EXAMPLE: A smart contract may be stored in a Trusted Execution Environment (TEE) or other security approaches may apply.

6.2.4 Home-PDL Managed Storage

6.2.4.1 Non-PDL External Storage

Contingent on availability of resources, governance can manage external storage. Smart contracts can be stored on an external storage if it meets the requirements listed in clauses 6.2.2 and 6.2.3.

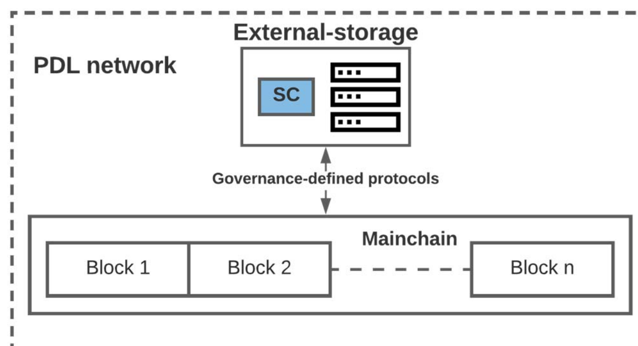

Figure 6-2: Smart contract is offloaded to a HPN managed external storage

6.2.4.2 HPN Managed Sidechain

When a smart contract is installed on a sidechain and is a part of the Home-PDL network, (that is, the sidechain is governed by the same governance as the mainchain) the following conditions shall be met, in addition to the requirement listed in clauses 6.2.2 and 6.2.3.

[RSC 1] The access control mechanisms shall be followed as per the same guidelines as in clause 6.3.7.

[RSC 2] If the sidechain is using a different PDL-type, the execution latency shall be compared with the mainchain.

NOTE: It is possible that difference in execution latencies between two PDLs, that is, mainchain and sidechain may produce ambiguous results.

[RSC 3] The parties responsible for this offloading shall ensure that results produced with offloading are valid and timely.

[RSC 4] Sidechain deletion or maintenance should not impact the smart contract functioning. That is, sidechain shall not be deleted or go under maintenance before all the offloaded smart contracts are moved to an alternate storage and functioning successfully.

[RSC 5] The governance/owner of the smart contract shall be notified well before for the sidechain planned maintenance that may impact the smart contract functioning.

[RSC 6] The notification time shall be part of the agreement between the governance/owner and the sidechain storage owner.

[RSC 7] If the sidechain is using a different PDL-type or version, governance/owner shall ensure that results produced are valid, timely and interpretable by the recipient chain.

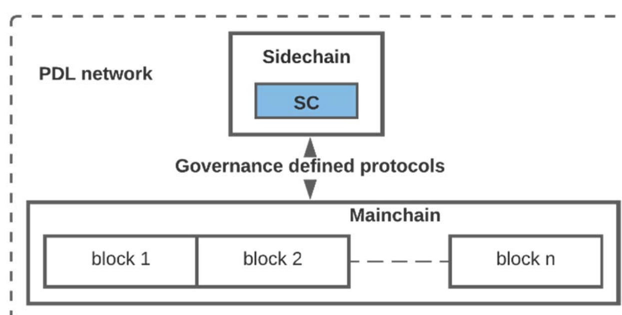

Figure 6-3: Smart contract is offloaded to a HPN managed sidechain

6.2.5 Non-HPN Sources Managed Storages

6.2.5.1 Introduction

When smart contract is managed by an external entity, which is not the part of the HPN, following can be scenarios:

- Non-Home PDL mainchain.
- Non-Home PDL sidechain.
- Third-party managed external storage.

The generalized ***requirements*** for scenarios in this clause are as follows:

[RFR 1] Governances of all the respective PDLs shall outline the access control mechanisms and oversee the sharing of the contract and the data.

[RFR 2] A verifiable agreement shall be completed and signed before the contract is offloaded.

[RFR 3] It is the responsibility of both the PDLs to ensure the integrity and reliability of the data accessed through smart contract.

6.2.5.2 Non-HPN Mainchain

Smart contracts can be installed on a foreign mainchain, that is a PDL which belongs to another PDL network. In Figure 6-4, PDL network 1 may need to access the smart contract installed on PDL network 2.

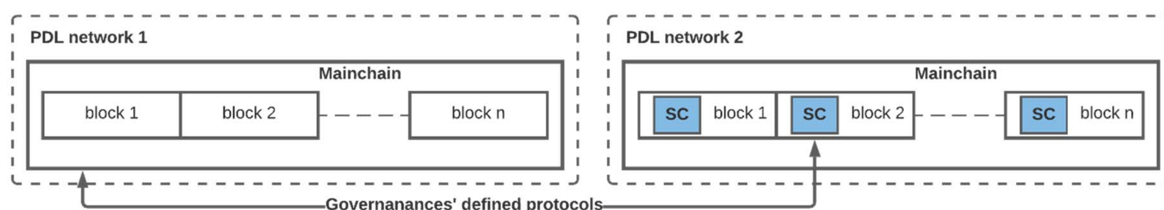

Figure 6-4: Smart contract offloaded to a Non-HPN mainchain

In such a scenario, following are the ***requirements***:

[RFMC 1] Smart contracts on a foreign PDL shall be accessed as per guidelines in clause 7.4.

[RFMC 2] Governances of both the PDLs shall define and manage the access control strategies (see details in clause 6.3.7).

[RFMC 3] Governance of the PDL network wish to share their smart contracts with other PDL network shall take consent from their participants before agreement to share.

[RFMC 4] The shared smart contracts shall not be updated/terminated/destructed without the consent of all the participating PDL networks.

[RFMC 5] The update/termination/destruction of the shared contracts shall be documented in a secure way.

[RFMC 6] The update/termination/destruction of shared contracts shall not be processed when any of the PDL is accessing the contract.

6.2.5.3 Non-HPN Sidechain

When a smart contract is installed on a foreign PDL sidechain, the requirements are same as listed in clause 6.2.5.2.

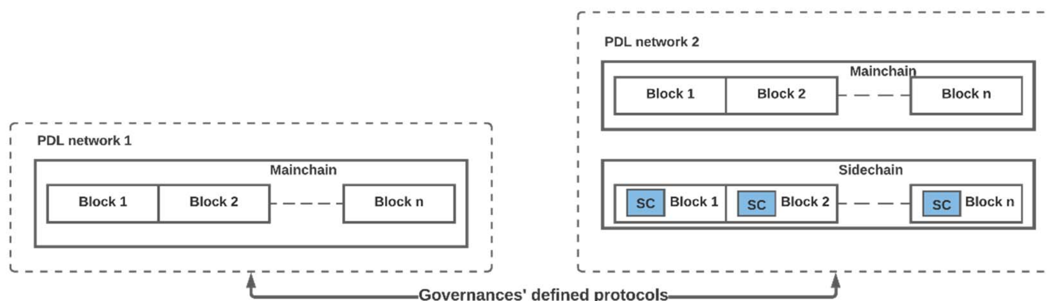

Figure 6-5: Smart contract is installed on a Non-HPN sidechain

6.2.5.4 Third-Party Managed External Storage

When a smart contract is managed on third-party storage. In additions to the conditions in clause 6.2.5.1, the following requirements shall apply:

[RTPES 1] Entity/party who is in control of the hardware and software shall be responsible for the security of the contract and its data.

[RTPES 2] The governance of the PDL shall ensure that third party storage follows security protocols which are compliant with the requirements of the application.

[RTPES 3] The governance of the PDL shall ensure that third party storage have, and they implement adequate hardware, software and security resources to install and manage smart contracts.

[RTPES 4] The party who is in control of the hardware and software shall be responsible for data stored and the inputs to the smart contract.

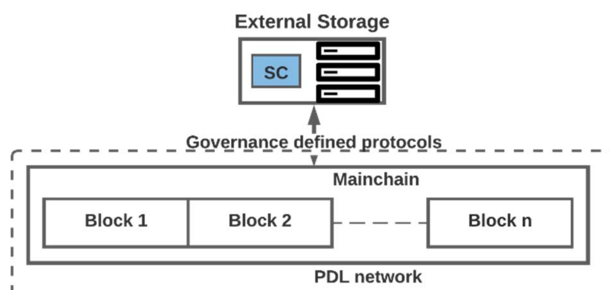

Figure 6-6: Smart contract on third-party managed external storage

6.2.6 Modularized Smart Contracts

6.2.6.1 Introduction

Smart contracts can be modularized into small components, called as ***Modularized Smart Contracts.***

In some situations, it may be feasible to split the smart contract components between different chains and storages. This is mostly useful to improve the performance as well as the security of the PDLs through segregation. For example, if a smart contract component is generalized and can be shared among several PDL networks, it may be feasible to install that component on a sidechain, so that it is shared/accessible among several PDLs without all the PDLs accessing each other's mainchain.

The general ***requirement*** for modularized smart contracts as follows:

[RDSC 1] All the smart contract modules shall work as a unit.

[RDSC 2] All the smart contract components shall avoid functional redundancies. That is, every component should have unique functions.

[RDSC 3] *Securely stored* - agreed by the governance and PDL consensus.

The general ***considerations*** for modularized smart contracts are as follows:

[CDSC 1] Management overhead caused by distributing the execution components.

[CDSC 2] Authorization and coordination may require additional resources such as additional time and coding.

[CDSC 3] Latencies may be different for different devices.

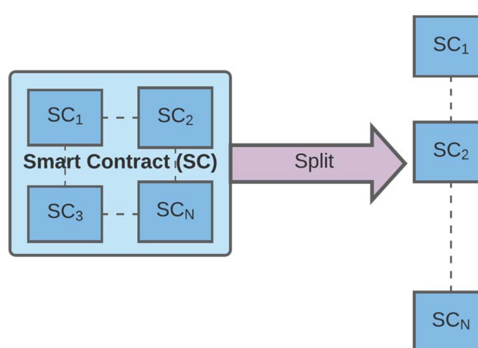

Figure 6-7: Smart Contract modularization into functional components

Following are the possibilities of smart contract functionalities distribution.

6.2.6.2 Sidechain Offloading

It is already demonstrated that a smart contract may be distributed into two or more different execution modules, for example, a code module which requires heavy computational resources and another component which calls/initiates those resource-incentive computations. In such a situation:

[OSCO 1] These execution components can be distributed among several off-chain and external storages.

This model is generally efficient in the situations when smart contracts may need to perform resource-intensive tasks such as, in future Machine Learning (ML) or Artificial Intelligence (AI) computations.

Following are ***requirements*** for sidechain offloading within the same PDL network:

[RSCO 1] The results produced by the functional components managed by the sidechain are timely and accurate.

[RSCO 2] The results produced by the functional components from a sidechain follow the same formatting as the mainchain.

[RSCO 3] It is possible that the sidechain is following different version and PDL type as the mainchain, in such a case, governance shall ensure requirements RSCO1 and RSCO2 are followed.

[RSCO 4] The difference is PDL type/version number shall not impact the operation of smart contracts and shall produce results as required.

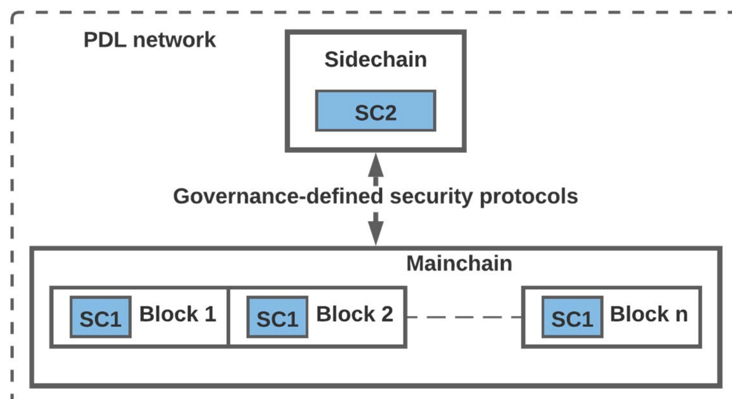

Figure 6-8: Smart Contract split into two execution components (i.e. SC1 and SC2) SC1 is stored on the mainchain and SC2 is stored on a sidechain

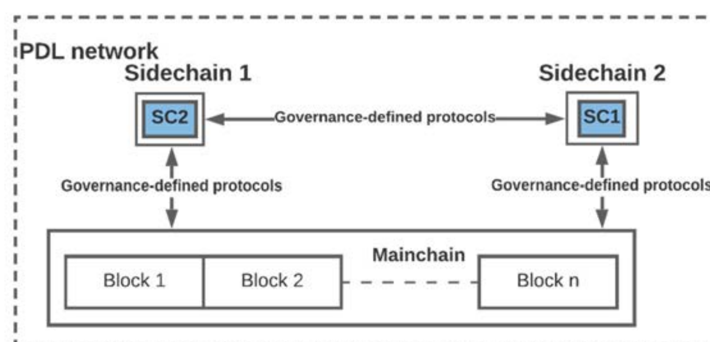

Figure 6-9: Smart contract execution split: SC1 and SC2 are stored in two different sidechains

6.2.6.3 HPN-Managed Datacentre

When a smart contract is stored in an HPN-managed datacentre, in addition to the conditions in clause 6.2.6.1, following are the ***requirements***:

[RHDC 1] Governance of the PDL is responsible for security and integrity of the data and datacentre.

[RHDC 2] The governance of the PDL shall ensure that results produced and/or executions performed on the external storage are timely and PDL compliant.

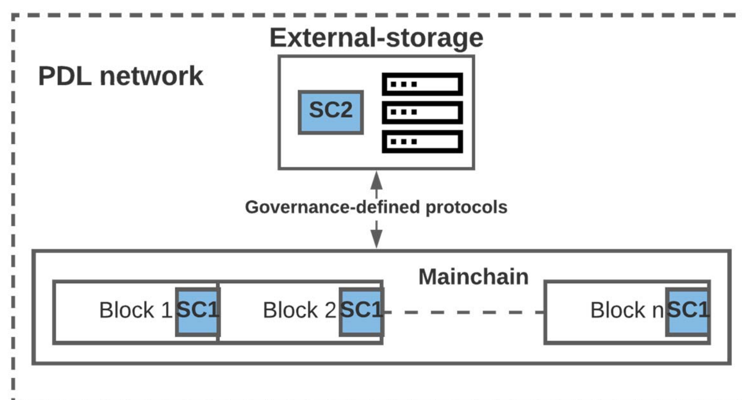

Figure 6-10: Smart contract execution split: SC1 is stored on the mainchain and the SC2 is stored on an HPN-managed external storage

6.2.6.4 Non-HPN PDL Smart Contract Split

6.2.6.4.1 Non-HPN Mainchain

In this scenario, smart contract execution components are distributed or shared among two or more PDLs.

[RSCS 1] The governances of the respective PDLs shall oversee the access-control mechanisms and outline the strategies to access the smart contract and the respective data (see clause 8 for detailed governance role in smart contracts).

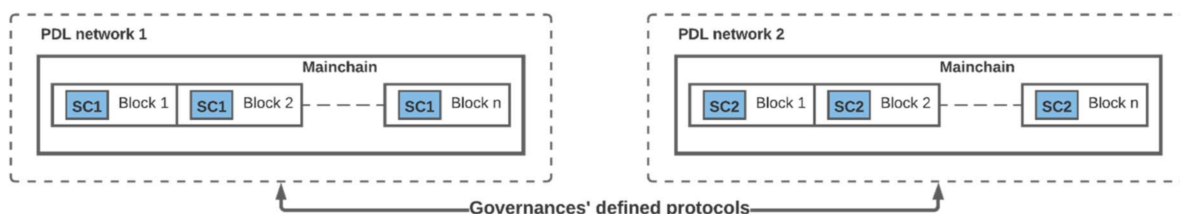

Figure 6-11: Smart contract split between two PDL networks

6.2.6.4.2 Non-HPN Sidechain

In such a case, requirement RSCS 1 applies.

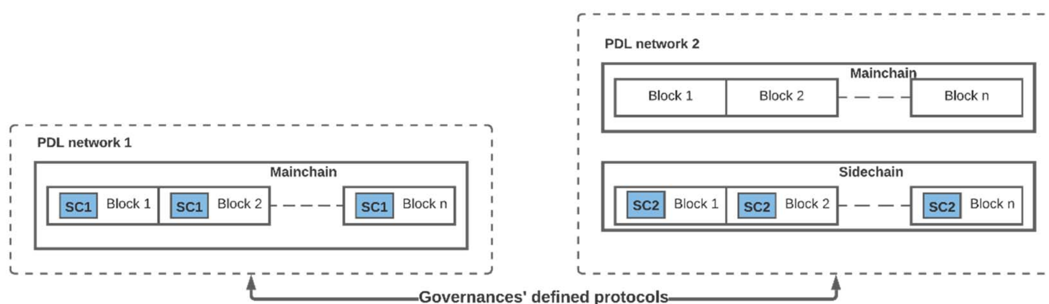

Figure 6-12: Smart contract is split between mainchain and sidechain of a non-HPN

6.2.6.5 External and Shared Storages Among PDLs

Clearly, it is also possible that a smart contract component is stored on an external storage and this block is shared among several PDLs.

[RESCS 1] The governance of the respective PDLs shall outline the access control and sharing strategies that are expected to be followed by the requesting PDL.

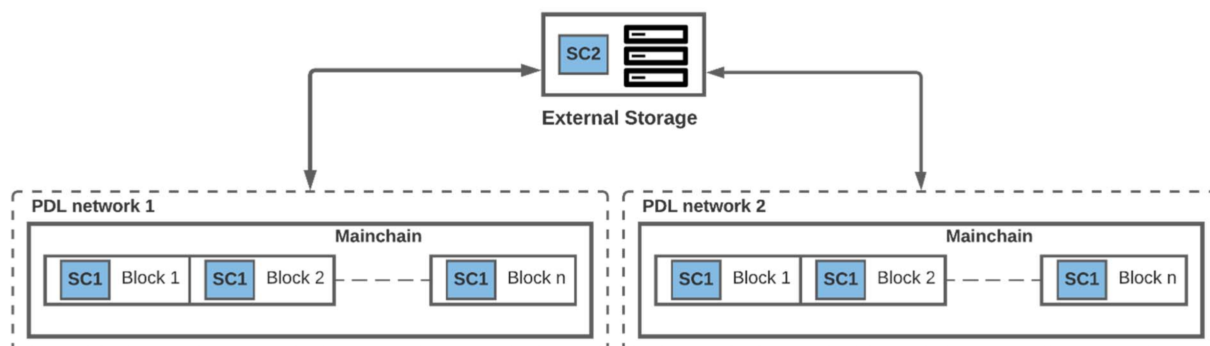

Figure 6-13: A Smart contract component is stored at an external storage and shared among several PDLs

6.2.6.6 Time-Dependent Smart Contracts

Often smart contracts are dependent on execution of other parts of other smart contracts, which may lead to synchronization and sequencing issues. In such cases, the executions may be stored in an interim data structure. Following are the requirements:

[RTDSC 1] If one component of a smart contract is executed faster than other components, execution results shall be kept secure in an interim data structure.

[RTDSC 2] The security of this interim data structure shall be managed by the operator in accordance with the governance-defined requirement for secure storages.

[RTDSC 3] The security of this interim data structure shall be audited by the governance of the PDL periodically as per the governance-defined guidelines.

[RTDSC 4] If the health of this data structure is not satisfactory, the governance shall take the necessary remedy actions.

[RTDSC 5] When a smart contract takes longer than other smart contract components, governance shall define a waiting time for the smart contract components to produce output.

[RTDSC 6] Operators of the PDL shall plan the executions considering the dependencies and smart contract (and data structure) latencies.

Considerations:

[CTDSC 1] In such a situation the subsequent smart contracts' execution can be invalidated because absence of the valid input (which was expected through a pre-requisite smart contract).

6.2.6.7 Smart Contract Time/Timers

In order to avoid the malicious or accidental uncontrolled executions of a contract, several timers may be a part of a contract and can be referred as 'Smart Contract Timers'.

The following requirements shall be followed:

[RSCT 1] A smart contract shall execute in a time-controlled manner, that is, owner/governance shall have control over the termination and interruption of the contract.

[RSCT 2] Smart Contract Timer shall be defined as a function of a smart contract.

[RSCT 3] When a smart contract is initialized, it shall also activate some internal functions, which can reset or destruct the contract to stop/interrupt the execution.

[RSCT 4] The smart contract functions shall be executed explicitly, hence the timer function shall be executed by the owner or governance.

[RSCT 5] All the timer functions included in this clause and other clauses throughout the present document shall maintain precision of nano seconds.

See clause 6.4.1. for details on required functions for a smart contract.

6.2.7 Data Integrity Sanity Checking

It is important that data entered to a smart contract is accurate, therefore the inputs can be verified for their validity at the middleware layer. Following are the ***requirements***:

[RDISC 1] Data entered to a Smart contract input data shall be accurate and timely.

[RDISC 2] The quality of the data shall be check at the middleware.

[RDISC 3] Data sanity checking is responsibility of the node/party executing the smart contract.

[RDISC 4] Governance shall take necessary measures to prevent nodes feeding inaccurate and invalid data.

[RDISC 5] When data sanity checker is implemented as a software, it should be configured to check the accuracy of the data thoroughly.

Options are as follows:

[ODISC 1] Data sanity checking can be implemented as software code.

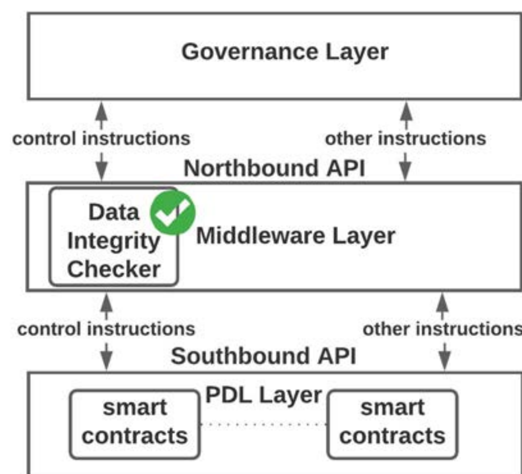

Figure 6-14: Data integrity shall be check at the middleware layer before the data is sent to the ledger

6.3 Architectural and Functional Requirements

6.3.1 Lifecycle of a Smart Contract

[RLC 1] All smart contracts shall follow the lifecycle stated in Figure 6-15.

[RLC 2] All the smart contracts will have terminations.

[RLC 3] The owner/governance keeps the rights to destructs the smart contract.

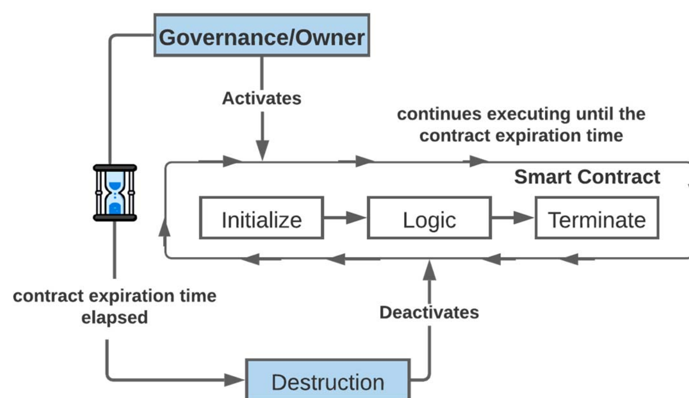

Figure 6-15: Smart contract lifecycle

6.3.2 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. To avoid that, following strategies ***can*** be applied to ensure safe and well-tested contracts:

[OTCO 1] It is recommended to use template contracts that have been tested and debugged to reduce the chance of such errors.

[OTCO 2] Template contracts are designed with more consideration and secured with several security checks.

[OTCO 3] Templates may require specification of the throughput/bandwidth of the contract and depend on the governance, application of the contract and the stakeholders involved.

6.3.3 Time-Limited

Eternal smart contracts can cause unwanted executions in future, Therefore, smart contracts shall have internal and external ways to limit their lifetime.

6.3.4 Internal Termination

Following are ***requirements***, that shall be defined as function of smart contracts to avoid eternal smart contracts:

[RIT 1] Smart contracts shall be terminated with an internal termination signal/command.

[RIT 2] Every smart contract should have a termination function that disables further executions of the smart contract after a certain period or upon notice.

[RIT 3] If a contract is generalized 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: Internal termination is a **short-term timer,** and a deactivated smart contract may be reactivated for future contract executions with different parameters upon signal or specific conditions.

Figure 6-16: Smart Contract Internal Termination

6.3.5 Safe Termination

Smart contracts can be assigned very sensitive tasks, so it is important to safely terminate them to avoid future attacks.

The ***requirements*** for safe termination are as follows:

[RST 1] Governance and owners of the contract shall ensure a safe termination of such order.

[RST 2] Ensure that none of the terminated smart contract functions is callable.

[RST 3] All the access rights of the current execution for the smart contract are revoked.

[RST 4] Data is archived for future reference such that data integrity should be maintained.

[RST 5] When a smart contract is required to be terminated, the reason of the termination shall be sent as an argument of the contract. Further details in clause 6.4.1.5.

Reasons for the termination **can** be as follows:

[OST 1] Natural Termination e.g. the purpose of the smart contract is fulfilled.

[OST 2] One of the parties is pulled-off the contract or revoked the contract (interruptive termination).

[OST 3] Timeout of the contract reached.

[OST 4] Other reasons such as malicious activities or other non-contractual actions.

6.3.6 Destruction

Destruction occurs when the long-term timer of the smart contract is elapsed, and the contract end date/time is reached. Destruction is different from termination (both natural and interruptive) of the smart contracts because a destructed smart contract cannot be reactivated.

Following are the ***requirements*** of a smart contract destruction:

[RD 1] Once destroyed the contract cannot be reinstated.

[RD 2] All the functions shall be inactivated before the destruction.

[RD 3] If a smart contract is required to be revised, that is, a new/revised version to be installed, it shall be classed as a termination, not as a destruction.

In a typical PDL, a smart contract cannot be initiated on its own, therefore, destruction shall be initiated:

[CD 1] The governance of the PDL can initiate the destruction through a control instruction (clause 6.3.8).

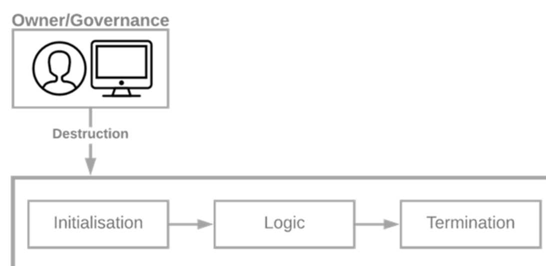

Figure 6-17: Smart Contract Destruction - Termination is different from the destruction and removes/deactivates the contract forever

EXAMPLE:

```
function destructContract(...) {
        if (endTime<=now) {
         Clear all variables and deactivate the contract completely}}
```

Code Extract 1: Smart contract destruct example

6.3.7 Secure Access Control Mechanisms

6.3.7.1 Introduction

In PDLs, smart contracts accessed by the PDL participants (e.g. owners) and external sources (e.g. off chain sources) through strict access-control mechanisms. Following are the ***requirements*** for accessing a smart contract.

[RSACM 1] Smart contracts should be accessible by the owner(s) of the contract, or any other party authorized/delegated by the governance.

[RSACM 2] The governance of the PDL shall ensure that access rights are granted to authorized members (internal and external) only.

[RSACM 3] Smart contracts access shall be in a time-controlled manner, that is every access to every smart contract shall have a limited-time access only.

[RSACM 4] Stringent access control mechanism should be implemented to enforce the same.

[RSACM 5] Governance shall maintain a record of access rights granted to parties/entities in an Access Control List (ACL).

[RSACM 6] The fields of an ACL **shall** include:

a) Node Identity (different from public key, and is assigned by the governance when the node joins a PDL)

b) Access start date

c) Access end date

d) Access start time

e) Access end time

f) Smart contract identity

g) Functions granted access to the smart contract

[RSACM 7] The access revocation shall be automated, and software based. After the agreed condition(s) are met the access rights are revoked automatically.

Options:

[OSACM 1] In some cases, the owner of the contract or governance may temporarily use **delegates** to assign rights of a contract.

6.3.7.2 Access - Control Requirements for delegates

[RACD 1] Access delegation should be for a limited time and will be revoked when such time elapses.

[RACD 2] If the delegation is changed, that is, a delegate further delegates the rights, 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.

[RACD 3] 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.

[RACD 4] 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.

[RACD 5] If a different device is identified than the authorized one than the access of the delegate shall be blocked, and warning should be issued.

[RACD 6] The delegation rights should be the function of the Governance Layer. 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.

[RACD 7] The delegate identification can be done through node identifier (Node Identifier) which is assigned by the governance to the nodes of the PDL.

6.3.8 Control Instructions

Though these are function which are executed in special circumstances, other than usual smart contract operations and are only for the purpose of control instructions. For example, sending an interrupt instruction. Some of the examples to send control instructions are as follows:

1) *Blocking/Unblocking Instruction* - if a smart contract is not performing as planned (because of reasons such as error in a contract).

   Such a contract should be stopped or blocked by exclusive instructions. Depending on the requirements of the contract, such instructions can be of several types. For example, invoke/execute the termination clause. See clause 6.4.1.5 for further details on termination functions.

2) *Function updates* - depends on the PDL-type. For example, at some instance, certain functions are mandatory to update/modify (see Table 6-1).

3) *Destruct Instruction* - As discussed in clause 6.3.6.

NOTE: Only the values of the functions can be modified, the functions, that is, the code is immutable.

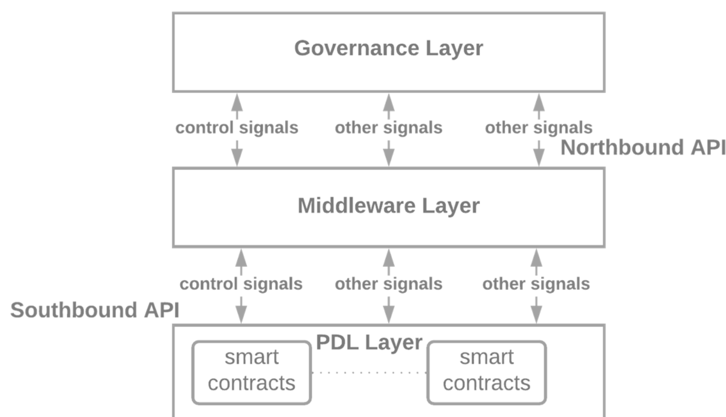

Figure 6-18: Instruction Flow Between the Layers

Following are ***requirements*** for a smart contract to include control instructions:

[RCI 1] Smart contract shall include instructions as functions to allow interrupt the execution.

[RCI 2] Following control instructions shall be defined in smart contracts:

a) Internal Termination (clause 6.3.4).

b) Interrupt Execution - that immediately stops the smart contract execution.

c) Destruct.

[RCI 3] Control instructions are of high importance and shall be accessible only by the governance and the owner of the contract with exclusive access control credentials.

Table 6-1: 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 (see note 4)	Yes	Yes	See clause 6.3.7.2
End date (see note 5)	Yes	No	No
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 duration. NOTE 3: The governance will delegate limited time duration. NOTE 4: Delegation should be the functionality of the Governance layer. NOTE 5: Can be used for extension.			

6.3.9 Archiving the data

Smart contracts are only execution code, the data generated by them is recorded in the ledger.

[CAD 1] 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 6.3.5) the contract.

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

Table 6-2

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 vendors 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:

[OAD 1] Back up the executions and data in a cold storage with a timestamp.

[OAD 2] Create a sidechain, and copy the executions there.

[OAD 3] Hash the smart contract and the respective data and store it in a local storage.

6.3.10 Stale data

If a smart contract is destructed or deactivated, the data generated by the smart contract will stay in the ledger for the lifetime of the ledger.

Smart contract logic and code shall be stored in a secure way within the ledger to keep the record of the operations performed on the data in the past (for example, strategies discussed in clause 6.2 for smart contract offloading).

6.3.11 Updates

See clause 10 for details.

6.3.12 Smart Contract Fields

A smart contract is initialized with some data, typically this data identifies the smart contracts. In a smart contracts two types of fields:

[RSCF 1] **Mandatory fields** - These fields shall be the part of a smart contract.

[OSCF 1] **Optional fields** - Owner/governance many introduce bespoke fields to serve the purpose of a smart contract.

6.3.13 Mandatory Fields

[RMF 1] Every smart contract shall include both fixed and parameterized fields.

[RMF 2] Developers/testers shall ensure that all the mandatory fields listed below are included in the smart contract before deployment.

6.3.14 Fixed/Permanent

Smart contracts are ***required*** to have the following fields. These fields are set at the time of the deployment.

[RFP 1] ***Contract ID*** - an internal identity of smart contract. This identity is allocated by the governance of the PDL. Following are fields shall be included in the ContractID.

ContractID = LedgerID:OwnerID:ContractIdentifier:

- Ledger Identity (LedgerID).
- Owner Identity (OwnerID).

- Contract Identifier.

NOTE 1: Contract Identifier is different than Contract ID.

NOTE 2: Contract Identifier is a unique identifier represents the contract within a PDL.

[RFP 2] ***Ledger ID*** - Every ledger shall be identified by a unique identity; all the regulated ledgers should be assigned an ID by the relevant authorities (such as governance or the owner).

Following fields may include in a Ledger ID, and the construct of the Ledger ID is out of the scope of this work:

- Region Identity (RegionID) (e.g. GB, IT)
- Company Identity (CompanyID) (Company or organization node in a PDL assigned ID) e.g. **RegionID:CompanyID**

[RFP 3] ***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 customized contracts for specific purposes. In both the cases, the owner ID should be the mandatory section of a smart contract.

[RFP 4] ***Start time (Governance-defined clock)*** - the start time of the contract, that is , time when the contract is deployed in the ledger and is different from the execution/invocation time of the ledger.

[RFP 5] ***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. Users of the contract shall ensure that all the sub-contract execution time is within the end-time of the contract (Figure 6-15). Depends on the scenario, it may be changeable, for example, an interrupt instruction may change this time to stop and revise the contract version.

[RFP 6] ***Version No.*** - Version Number of the smart contract, if a smart contract is re-deployed after termination, same versioning sequence shall be followed.

6.3.15 Parameterized

[RPR 1] ***Execution Start Time*** - start time of a particular smart contract execution.

[RPR 2] ***Execution End Time*** - end time of a particular smart contract execution. It shall be before the contract end time.

[RPR 3] ***Execution ID*** - identity of execution by a user of the PDL or external entity.

[RPR 4] ***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. Anonymization should be resolvable to ensure accountability in PDLs. Pseudo-anonymized by the PDL governance may be considered.

[OPR 1] In special or exception circumstances (e.g. malfunctioning of the contract), some of the fields may be updated with the discretion of the governance.

6.3.16 Optional Fields

Optional fields of a smart contract depend on several reasons such as purpose, usage and timing. It is up to the governance and the owners of the contract to introduce optional fields as per their requirements. These fields are out of scope of this work; however, some examples are listed below:

- Smart Contract genre description.
- Corresponding paper contract reference numbers/identifiers.
- List of middle-parties involved in the contract.

- Corresponding country/region laws if applicable.

[ROF 1] These fields should be clearly identified as optional fields within the contract. That is, within the contract it should be clear that the field is optional or mandatory.

[ROF 2] Mandatory fields should be maintained in a container field, which will maintain all the optional fields.

6.4 Architecture and Functional Descriptions

6.4.1 Required Functions

6.4.1.1 Initialization

The contract is initialized with the initialization function that is the constructor of a contract and prepares the smart contract for executions such as initialization of the variables.

The initialization function:

[RINT 1] Shall include initialization of variables.

[RINT 2] Initialization function shall be checked explicitly, along with logic functions at the testing stage (from the lifecycle [i.1]) because if not present, the smart contract can be dormant.

[OINT 1] May call to logic functions of a smart contract. Initialization functions may also verify the login credentials.

In Figure 6-19, the key elements of a smart contract are shown.

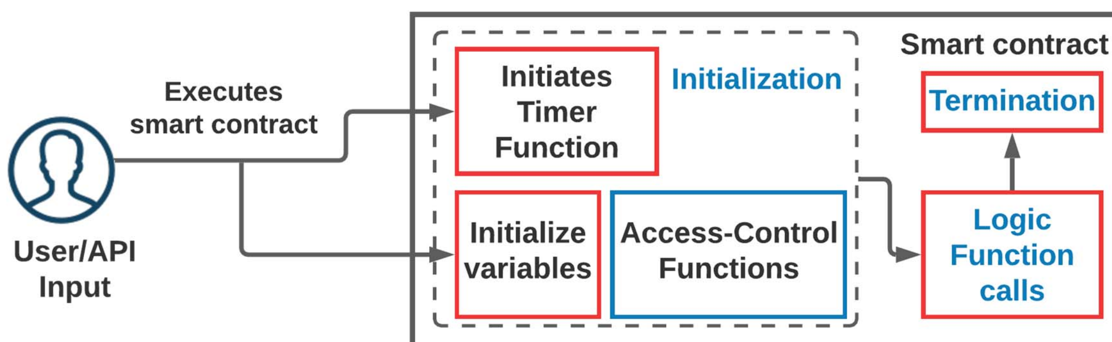

Figure 6-19: Initialization Elements of a Smart Contract

The Timer of the smart contract is initialized whenever the smart contract is executed. In some cases, the timer may be initialized with the first execution of the smart contract and keeps running (possibly pausing from time-to-time on certain conditions) until the contract is deactivated.

Some of the conditions for pausing/deactivating are listed below:

- Achieved the lifetime of the contract.
- Once execution is completed (but the contract can be executed in future).
- Malfunctioning of a smart contract.
- Access rights given to malicious users accidently or otherwise.

Timer function usually calculates the time elapsed relative to local time:

```
function timer_function () {
    Time_elapsed = now () +nano_seconds_elapsed;}
```

Code Extract 2: Example of a timer function

[RINT 3] **now ()** in Code Extract 2, is the governance time.

[RINT 4] Nodes shall maintain all the timers to nano seconds precision.

6.4.1.2 Access-Control

The Access-Control Function inside the smart contract will ensure that users who are executing the smart contract functions are authorized to access those functions.

[RACF 1] In addition to Access Control of a smart contract (clause 6.3.7), stringent access control shall be the part of the smart contract itself as a function.

[RACF 2] Access control functions shall verify if the user has sufficient rights to access a particular function.

[RACF 3] Access Control functions shall be hardcoded and grant/block access by checking the role of the caller.

[RACF 4] Developers should design a smart contract to ensure that Access Control Function is not callable by any other function (within or outside smart contract).

EXAMPLE: In an auction contract, bidders can place a bid in the auction but may not be allowed to stop it. Therefore, they will not be allowed to access the endAuction() function of the smart contract.

6.4.1.3 Logic

Logic Functions are the main tasks of a smart contract.

[RLF 1] Logic functions shall be accessible by strict access control mechanisms as discussed in clause 6.4.1.2.

[OLF 1] They may include several classes and functions which do the executions to serve the smart contract purpose.

6.4.1.4 Entry Functions

[REF 1] In a typical case, contract users shall be able to access only a certain subset of functions.

[REF 2] Interaction between smart contracts' functions should be limited and controlled to prevent unauthorized access to other users' data.

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.

[REF 3] Other functions (as an example above) should only issue the payment and shall not invoke/initiate any other function of the contract, for example, historical data of payments.

[REF 4] By default, Initialization Functions (clause 6.4.1.1) are Entry Functions because they are the entry -point for any outside request. However, other smart contract functions accessible by Oracles/APIs shall also be classed Entry Functions.

[REF 5] Developers/Testing Engineers shall test and verify that no backdoor is present from entry functions to other functions.

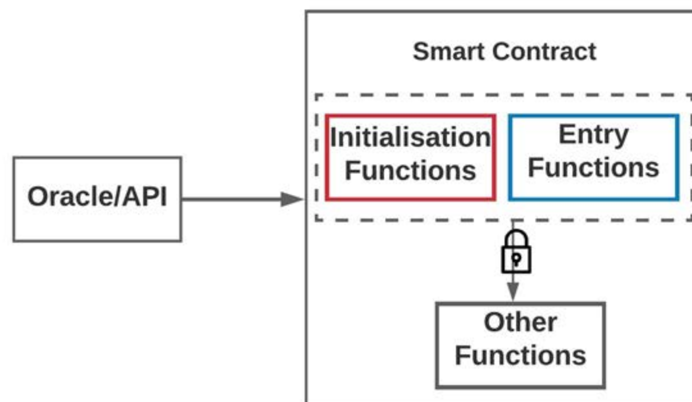

Figure 6-20: Entry Functions shall not create back doors to other functions of the contract

6.4.1.5 Termination Function

[RCT 1] As explained in clause 6.3.3, smart contracts shall be terminated at a certain point when certain conditions are fulfilled, to avoid future accidental executions. A Function that performs the termination of a smart contract shall be present in the smart contract.

[RCT 2] The termination function shall be accessible by the governance/delegate or owner of the contract with appropriate access rights.

```
function terminateContract(str reason …) {
if (termination_condition == True) {
Clear all variables and revert access rights}}
```

Code Extract 3: Example of Termination Function

[RCT 3] In case of interruptive terminations, for example, termination before the end of the contract, a Smart contract shall not be terminated without the agreement of all the stakeholders and the governance.

[RCT 4] In both Interrupt Termination and Natural Termination, reason of termination shall be sent as an argument with the termination function. Also, participants may try to send invalid reasons such as *xxx*.

[RCT 5] Governance of the PDL shall ensure that only valid inputs are sent to the termination functions.

[RCT 6] If any participants identified sending invalid string as argument, governance shall take necessary measures of compliance such as blacklisting of the nodes (node reputation).

[RCT 7] Unmeaningful reasons, that is the reasons that do not clearly identify the problem of the interrupt shall be categorized as Invalid Reasons.

[RCT 8] All the reasons shall provide the details of the interruption.

[RCT 9] Brief reasons shall not be used, that is, few words description shall not be used, and the reasons shall provide the complete description with identifiers.

[RCT 10] The reasons shall provide complete information specific to the jurisdiction and the contract.

[OCT 1] To avoid problems of compliance such as invalid arguments in the function (RCT 5), the governance may restrict the termination to themselves.

Examples of Invalid Reasons

Some of the examples for invalid reasons are as follows:

- **Invalid version** - invalid version is not a valid reason, details such as why the version is invalid should be included.
- **Human error** - shall specify the type error for example, typo with missing information along with corrections.

EXAMPLE: Wanted to send ABC as input but sent AB only. The correct input is ABC.

- **Example corrected version** - should also include the date and type of error such as Person A has activated the contract with wrong parameters.
- **Missing information** - specify the missing fields.

Example of Valid Reasons

- Smart contract completed life cycle.
- New version of the contract is required.
- Smart contract completion.

6.4.2 Example Architecture of a Smart Contract

6.4.2.1 Smart Contract with External Input

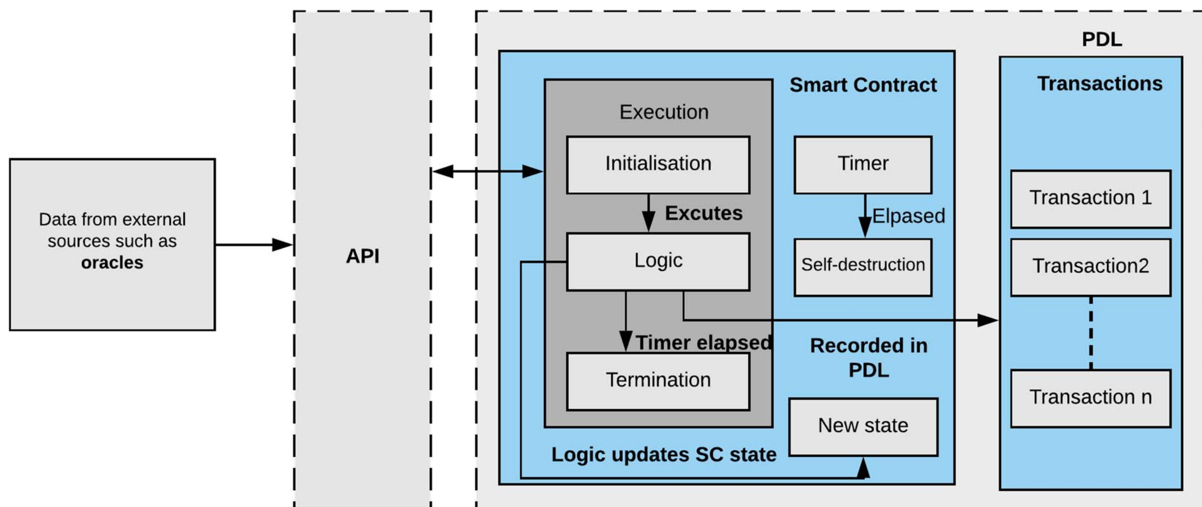

Figure 6-21: Reference Architecture of Smart Contract with External Input

6.4.2.2 Smart Contract with another smart contract

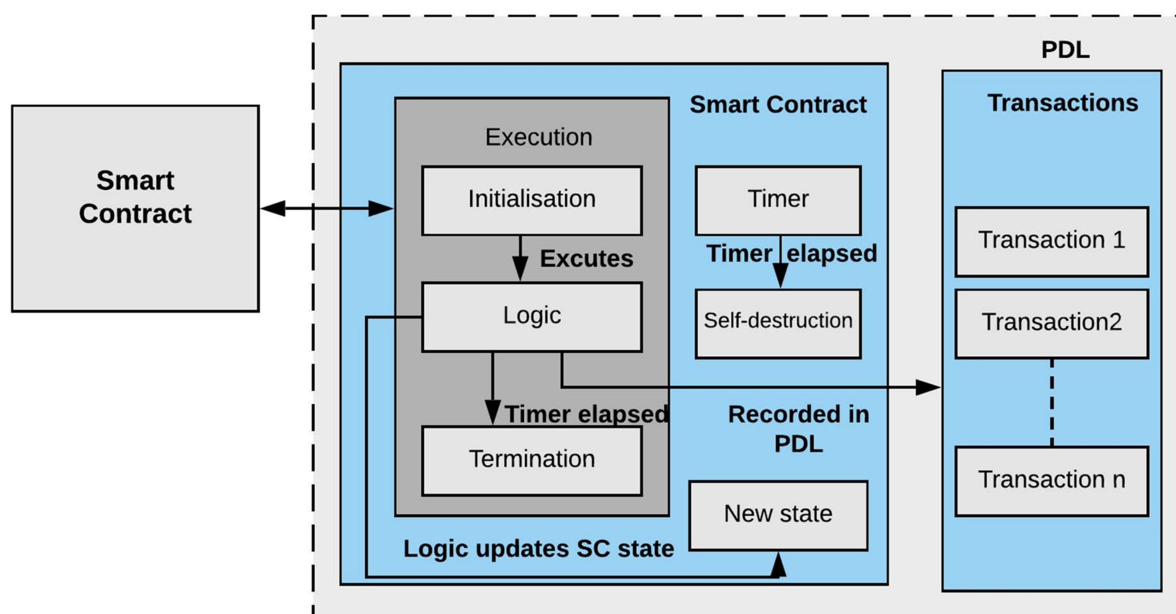

Figure 6-22: Reference Architecture of Smart Contract with Internal Input

7 Data Inputs and Outputs

7.1 Introduction

When interacting with the environment, smart contracts may take input from internal and external data sources such as other smart contracts and webservers. They may also provide output to other sources such as local and foreign PDLs and webservers.

Several combinations of data inputs and outputs are possible. A smart contract may take inputs from:

- Another smart contracts (internal or external).
- Oracle services.
- Other sources such as webservers and databases.

And may provide output to:

- Other smart contracts (internal or external).
- Oracle services.
- Other sources such as webservers and databases.

7.2 Generalized Input/Output Requirements

Smart Contracts do not have any built-in mechanisms to verify the integrity of the data. Therefore, it is important that the data input to a smart contract is trustworthy. The generalized **requirements** for the data inputs are:

[RINP 1] ***Integrity:*** Data shall be untampered and unaltered. It is forbidden to send altered or tampered data.

[RINP 2] Governance shall take punitive actions against entities that enter/or try to enter altered or tampered data.

[RINP 3] ***Accuracy:*** The data input shall be accurate and trustworthy.

[RINP 4] ***Quality of the data (e.g. syntax, semantics, context):*** A smart contract shall be given accurate input. Parties executing the smart contract shall ensure that they send accurate and timely input to the smart contract.

[RINP 5] ***Security:*** The inputs are secure from attacks such as the "Man-in-the-Middle Attack". This can be dangerous in some use cases such as in an auction contract if a bid value is intercepted, can affect the validity of the auction.

[RINP 6] ***Authenticity:*** Data shall be from the authorized users with appropriate access rights only. The allocation of access rights is discussed in clause 6.3.7.

[RINP 7] ***Sequencing and synchronizing 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. In such a case, it is important that the execution of the pre-requisite shall be completed before hand.

7.3 Internal Data Inputs

7.3.1 Introduction

7.3.1.1 Internal Inputs Options

Internal data inputs are the inputs from the HPN. A smart contract may take input from:

- HPN participants.
- HPN smart contracts.

7.3.1.2 HPN Participants

Typically, HPN participants access smart contracts through a transaction request. In addition to the requirements specified in clause 7.2, following are the requirements:

[RIPINP 1] The Smart contract shall receive data from authorized participants only.

[RIPINP 2] The participants sending the data/parameters to smart contracts shall ensure that data is accurate.

[RIPINP 3] The access control mechanism shall be handled through identity and permission control services defined in PDL Reference Architecture (ETSI GS PDL 012 [i.3] Work in Progress).

NOTE: Selected internal PDL participants can access smart contracts within the same PDL with access rights allocated by the governance and owner of the contract and as per guidelines stated in clause 6.3.7.

7.3.1.3 HPN Smart Contracts

When a smart contract is dependent on the data from another smart contracts' output, the important challenge is the completion of pre-requisite contract. In addition to the requirements specified in clause 7.2, following are the requirements:

[RSCINP 1] The pre-requisite smart contract shall be completed before sending the subsequent request.

Only some fields and functions of a smart contract are accessible by other smart contracts.

[RSCINP 2] Developers shall ensure that there is no ambiguity in access allocation.

[RSCINP 3] Execution triggered by unauthorized smart contracts shall be rejected.

The unauthorized access will be granted by chain of the contracts due auto-execution property of smart contracts.

NOTE: Selected internal PDL smart contract can access other smart contracts within the same PDL with access rights allocated by the governance and owner of the contract and as per guidelines stated in clause 6.3.7.

7.3.2 Requirements for Internal Outputs

A smart contract may provide output to other smart contracts within the PDL network. In such a case following are the requirements:

[RIO 1] The data produced by smart contract shall follow the requirements specified in clause 7.2.

7.4 External Data Inputs

7.4.1 Non-HPN

7.4.1.1 Introduction

A smart contract may take input from foreign PDL networks, external participants or smart contract. In such a scenario, following are the **requirements:**

[REPINP 1] The foreign PDL governance shall manage access of their respective PDL's smart contracts and assign access-rights accordingly.

[REPINP 2] Appropriate access-rights shall be acquired before such access.

[REPINP 3] External data shall be exchanged or accessed through the governance-channel only and participants from any side shall not have any direct access to the foreign PDLs.

Two possible approaches to access the data from an foreign PDL as are follows:

- Faster Approach.
- Secure Approach.

7.4.1.2 Faster Approach

When PDL participants want to access the data from a foreign PDL, they can generate a request-to-access message to their local governance which will subsequently send this request to the *Gateway*.

The Gateway can be maintained and administrated by both the internal and external governance and elected by mutual consensus of all participating PDL governance.

[REFAINP 1] All the access from an foreign PDL **shall** be recorded by the Gateway in a separate secured storage as well for future audit.

In Figure 7-1 the possible architecture for external data access is shown. The Gateway maintains an access control list, that keeps the record of the data shared between the PDLs. Note that, this access control list is also maintained in secured data structure. This is a faster approach but have some security considerations which are listed below.

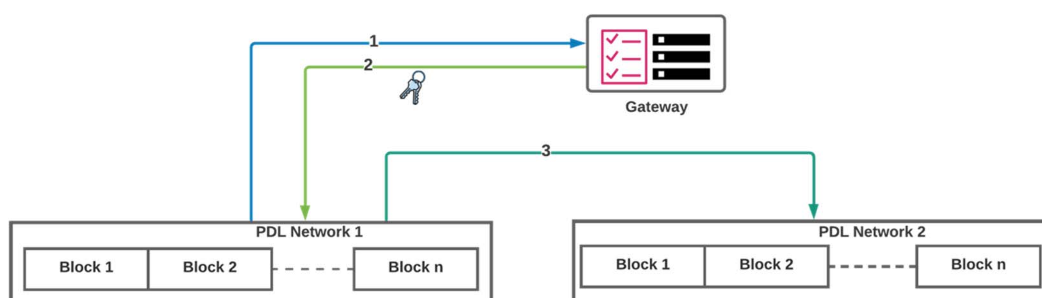

Figure 7-1: Smart contract from foreign PDL/Non-HPN (Faster Approach)

The ***considerations*** with this approach are as follows:

[CEFAINP 1] The major security consideration here is the single point of failure for the Gateway. This means that if the Gateway is compromised, the malicious party can take over the system and issue the keys to themselves or possibly to other malicious parties.

To resolve such a vulnerability, a secured approach is discussed in the next clause.

7.4.1.3 Secured Approach

To avoid single point of failure as discussed in the last clause a secure approach is to request the access rights in a dynamic manner (Figure 7-2).

In the secured approach, when PDL participants generate request-to-access to their local PDL governance, the governance forwards this request to the Gateway. The Gateway sends the access request to the governance of the foreign PDLs. Upon approval access is granted or rejected. Note that, in this case there is no access list maintained by the gateway and decision to grant/reject access rights are made dynamically. However, the gateway does maintain a list of access granted by the PDLs for future audit.

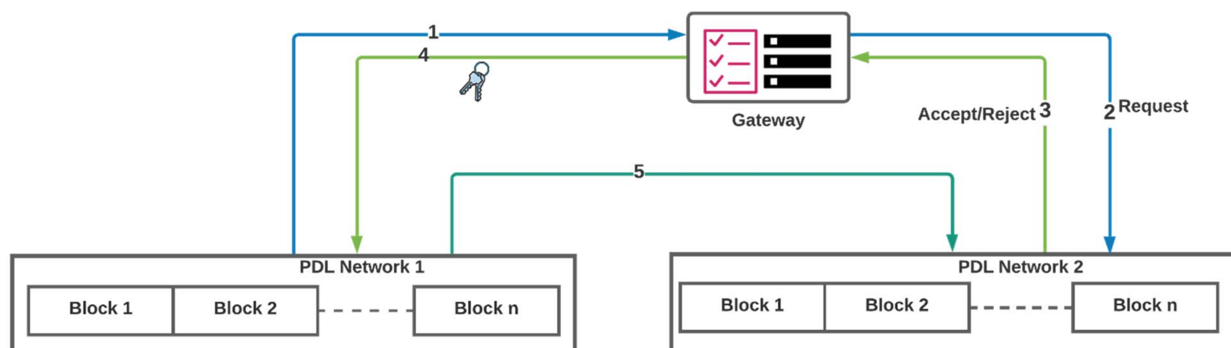

Figure 7-2: Smart contract from foreign PDL/Non-HPN (Secure but slower approach)

7.5 Oracles

7.5.1 Introduction

Smart contracts often interact to the outside environment, that is, in many situations, they may take data from external sources (e.g. marketplaces and weather data). They may also send data to the outside world. A common medium is the "Oracles". Oracles extract and verify data inputs for the smart contracts and PDLs. They also send the data to the outside world from the PDL in the similar manner. Typically, oracles are services (e.g. web APIs) which extracts the data from external data sources and translates it into PDL-understandable format.

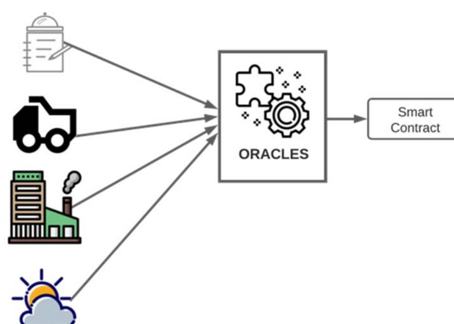

Figure 7-3: Oracles take input from several data sources processes them and provides input to smart contracts

Generally, oracles are divided into two different categories:

1) **Hardware oracles:** Many applications of PDLs require data input from hardware probes such as thermometers, pressure gauges, distance measurement devices. Some of these hardware devices generate analogue output that needs to be converted to digital using A/D convertor. A/D convertors have an intrinsic element of error.

2) **Software oracles:** The input sources for the oracles are software such as stock exchange data from their website/WebAPIs. This data is digital to begin with, so no conversion is needed, and the above element of error is eliminated.

NOTE: In both cases the oracles should be programmed to convert the information into meaningful data format for the purpose of the PDL application.

Oracles are blockchain-agnostic and typically generalized enough to access any PDL and provide the data. The problem with oracles that problem is that they are sometimes not trustworthy.

7.5.2 Requirements for Oracles

To avoid delay in transaction processing Oracles shall have the following properties:

[ROS 1] **Availability** - Governance shall implement strategies to ensure the availability of the oracles. That is, when a smart contract needs the data, it shall be pre-processed by the oracles and available for the smart contract input.

Data such as stocks and weather are time-dependent (vary with time) and change frequently. When a smart contract needs such data through an oracle, there may be some discrepancy due to delay in data generation, extraction and execution time and the latency of smart contract itself. That is to say, smart contracts may be slower in execution than the speed of the data generated which may result in getting outdated data.

[ROS 2] **Security** - Provenance and channel - Governance shall implement strategies to ensure the source of the data is trustworthy and the channels are secured to prevent interception such as man-in-the-middle attack.

[ROS 3] **Governance Approved Oracles list** - In the case of external oracle services, they **shall** be authorized by the governance. That is, governance **shall** have a list of approved oracle services which may provide the data to the PDL.

[ROS 4] **Trustworthiness** - PDL governance shall ensure that the malicious oracles cannot provide data to smart contracts. Oracles may process the data from several other sources such as, for the stock exchange information, the oracles may take data from several stock exchanges and processes them before providing it to the ledger. Generally, there are several websites/APIs publish this data. This compromises the integrity of the data. To tackle with this problem, oracle services **shall** only take inputs from the verified data sources.

[ROS 5] **Performance -** Oracles **shall** have performance matching to the PDL. For example, some PDLs have high Transaction throughput. The oracles shall match the performance or throughput of the PDLs, to ensure the availability and timely data. Oracle services shall also ensure that they can cope with the demand of the PDL data, for example, some PDLs may request large amount of data. Oracle services shall ensure that they can match with the volume demand. Oracle services shall be scalable such that they can cope of with inflated demand of the requests from a PDL.

7.5.3 Oracles' Access Rights

[ROAR 1] The governance shall assign the access rights for the oracles and for the required duration only. That is the, the start and end dates should be explicitly defined in the request to access the data.

7.5.4 Oracles as Internal Service

[OOAR 1] If the PDL participants have the resources available, they may run their own oracle service. This may ensure the authenticity of the data.

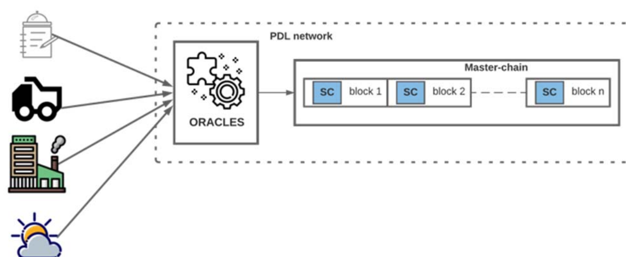

Figure 7-4: Oracle service internal to the PDL network

7.5.5 Oracle from External Sources

[ROES 1] Governance shall take sufficient measures (depending on the use case) to ensure the integrity of the service and the data.

[OOES 1] If an oracle service provider is misbehaving/faulty (e.g. providing wrong data), the governance may blacklist/suspend the oracle service.

[OOES 2] PDL participants may outsource the oracle service to other entities.

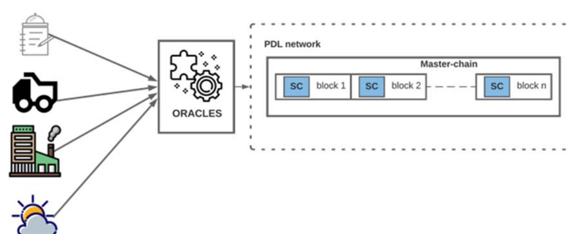

Figure 7-5: Oracle service external to the PDL network

7.5.6 Criteria for Oracles Services Approval

The governance of the PDL will register the services of oracles through application procedures. Application procedures are up to the preferred methods of governance. For example, governance may take oracles services applications through an open call or specific invite. Yet, the approval requires mandatory requirements listed below. Note the difference between the application procedure, which is the governance-defined method, and the approval procedure shall follow the following guidelines.

[RCOA 1] Smart contract shall have a built-in mechanism to verify the validity of oracle.

[RCOA 2] The list of approved oracles shall be updated periodically. This is the governance discretion to set the timeline of this review, but the review period shall not be more than three calendar months.

[RCOA 3] Governance shall define the criteria of the approval before or at the time of the PDL formation. This shall include:

a) Oracles shall be from authorized and validated sources only.

b) Time-limited Approval- approve the oracles for a certain governance-defined time only. After this time, the access shall automatically be revoked and come under review.

c) Oracles services shall be using transparent and auditable software to translate the data.

d) The oracles services code and architecture shall be auditable without any additional monetary transaction.

e) The oracles services shall be transparent.

f) Oracles services shall be using government-approved software libraries only.

g) The author/source of data shall be specified by the oracles before/at the time of approval.

h) Oracles shall be using authorized sources only. That is, they shall sign IPR agreement with the data sources before making application request with PDL governance.

[RCOA 4] Oracle sources shall provide details of their software and hardware security methods for data protection and authentication to the oracle services.

[RCOA 5] Oracles services shall maintain an online open directory for their available oracles.

[RCOA 6] All oracles shall be verifiable through this open directory.

NOTE: These guidelines are in the scope of generalized governance and PDL roles. The details including those are listed above are included in ETSI GS PDL 012 [i.3] Reference Architecture.

7.5.7 Offline Oracles

An oracle (either external or internal) could become offline or unavailable due to many reasons such as the loss of communication connection.

Mechanisms are required to guarantee the availability of oracles.

EXAMPLE: A smart contact can be coded or provisioned with the addresses or APIs of multiple oracles; when one oracle becomes unavailable, other oracles can be used by the smart contract.

8 Governance Role in Smart Contracts

8.1 Introduction

Generally, Governance of a PDL oversees the overall operations of the PDL. This includes but not limited to access control and operational strategies. Yet, the role of governance depends on the consensus of the PDL founding participants. Governance can be automated as well, in which a software program can take the decisions based on the pre-programmed conditions.

In this clause, the governance role specific to smart contract is outlined and general role governance role is out of the scope.

8.2 Governance Role Delegated to Policy of Smart Contract

Governance of a PDL **shall** take the following decisions for smart contracts:

[RGR 1] Listing the approved oracles for smart contracts.

[RGR 2] Listing the test strategies for smart contracts.

[RGR 3] Identification and acceptance of new members for a smart contract.

[RGR 4] Updating a smart contract.

[RGR 5] Access to smart contracts from external sources shall be contingent on approval of both internal and external governance.

Governance of a PDL **may** take the following decisions for smart contracts:

[OGR 1] Transaction approval time - a time smart contract shall wait to for its next executions.

[OGR 2] Installing a smart contract - a new smart contract is installed with the governance's approval.

8.3 Updating a smart contract

[RGUP 1] Governance's approval shall be required to update a smart contract.

A smart contract update can be required in several scenarios listed in clause 10.

8.4 Operational Decisions

[RGOD 1] Operational decisions are the responsibility of governance, and the owner of the contract shall follow the governance's advice to ensure the successful operation of a smart contract.

The operational decisions a **governance shall** involve in are listed below:

[RGOD 2] Start date/time and end date/time of the contract.

[RGOD 3] Allocate access rights to all the actors with in the PDL network.

[RGOD 4] Updating the contract.

[RGOD 5] Contract versioning.

[RGOD 6] Approved software/hardware technologies.

[RGOD 7] Access control strategies, technologies, and algorithms.

[RGOD 8] The external participants who can access the contract - this is important because the owner shall not allow the external entities to access the contract if not allowed by the governance.

[RGOD 9] Allocate unique identities to all the actors (clause 5.2) with in the PDL network.

The **owner shall** be responsible to decide on:

[RGODO 1] The internal participants who can access the contract.

[RGODO 2] Choose between governance approved technologies.

[RGODO 3] Testing strategies listed in governance guidelines.

[RGODO 4] Oracle's list approved in governance guidelines.

8.5 Termination of contract

Smart contract termination is a critical event that may affect (e.g. adversely) the behaviour and content of the PDL. Following are the requirements for smart contract termination:

[RGT 1] Owners of the contracts shall not initiate termination without the agreement of the governance.

[RGT 2] All the stakeholders and the governance shall approve the termination before the termination is initiated. That is, participants of the contract shall not pull out of the contract by terminating it.

[RGT 3] Participants shall take the full responsibility of the smart contract and ensure the termination follows a standard procedure.

[RGT 4] When it is identified that a smart contract is not working as required and needs to be terminated the governance shall be informed of this problem without any delay.

[RGT 5] In case of malfunctioning, after the governances' consent, if the smart contract is still active it will be turned off.

[RGT 6] If a revised (updated) smart contract is required, it will need to be activated either before or after the termination of the previous contract as applicable.

[RGT 7] If it is not active and/or does not require a replacement (e.g. dormant), the governance shall terminate the contract without a revised smart contract.

8.6 General Compliance Strategies for Smart Contracts

PDL governance may follow the following guidelines to make a smart contract secure:

[RGC 1] Compliance measures are dependent on the local laws. The PDL governance shall ensure that a contract follow the laws in the respective juridical.

[RGC 2] In case of cross-border PDL, that is, a PDL network where many governance laws are involved, the governance of the PDL shall outline the strategies of laws the coded in smart contracts, when initiating the PDL and as per the laws applicable for such scenarios.

[RGC 3] The penalties is the governance and PDL founders decision. But all the compensation and penalties shall be recorded in a document and signed by all the parties at the time of PDL initialization.

[RGC 4] The PDL participants that join the network later, shall made aware of the document and its contents and sign the document before joining the PDL.

[RGC 5] Any wrongdoing detected after the damage the governance may blacklist/ block the node. Penalties/compensation can be imposed on the malicious nodes.

[RGC 6] Step wise approach - first offence, second offence and so on. At the end node can have lifetime ban or high penalties.

9 Testing Smart Contracts

9.1 Introduction

Testing for any software program is an essential step before its deployment. Rigorous testing can prevent errors and enable designing safe and correct smart contracts.

[RT 1] Smart contracts can be tested like any other software, but an additional layer of testing shall be applied. That is, to ensure a smart contract interaction within itself and external entities (inter-PDL and intra-PDL) is protected through rigorous access control mechanism at the governance and smart contract layers.

For the sake of simplicity, the present document is focused on the testing strategies specific to smart contracts rather than generalized software testing mechanisms.

Testing is an important indicator for the unintended smart contract behaviour and shall ensure the following:

[RT 2] **Modularity** - If the test fails, the smart contract shall clearly indicate which part of the test failed.

[RT 3] **Well-structured** - The test codes should have clear indicators of the errors. For example, instead of generalized term such as Exception, a more specific exception type (e.g. IntegerOverflow Exception) shall be used.

[RT 4] **Clear and self-explanatory** - The tests shall use meaningful variable names; this will assist future debugging.

[RT 5] **Parameterized** - A testing code shall be parameterized, that is, it shall allow test engineers to pass a wide range of parameters to validate and verify the smart contracts' behaviour with different data types.

Typically, smart contracts shall test in two stages:

[RT 6] **Unit Tests** - Tests small units or functional blocks of a smart contract.

[RT 7] **Integration Test** - Compile or add all the functional blocks and test the complete end-to-end smart contract.

9.2 Testing Strategies

The smart contract shall follow the security protocol listed in the present document. That is, it shall be secure and impenetrable. However, to achieve this, developers can follow test strategies best suited to them.

[RTS 1] Testing targets listed in clauses 9.1 and 9.4 shall be met.

[RTS 2] Modular and Reusable - smart contract tests are designed in a modular and reusable fashion. These modular tests are managed by the governance and can be shared among the participants of the PDL. This can be useful, as it will ensure the reusability of the tests and may save time to design specific tests.

[RTS 3] It is up to the developers to design and implement test strategies. However, the resultant smart contract shall be secure and efficient, that is, it shall follow the guidelines listed in clause 6.

Some of the testing strategies developers can adopt are listed below:

[OTS 1] Automated Testing - some of the tests may be automated, that is, a smart contract can be verified through automated engines (for example, Remix), that may be able to verify some traits of a smart contract, for example, presence of certain required libraries.

[OTS 2] Outsource testing - In some cases, the governance of the PDL may prefer to outsource the testing procedure. In case of outsourcing testing, it is required that the testing firm meets the standards and follows the same procedures as listed in the present document.

[OTS 3] To avoid several branches and conflicts at the end, the developers may choose to use Continuous Integration and Continuous Delivery (CI/CD) technique. In this technique, a small code is written and integration to avoid the conflicts. In the situations, where enhanced testing is required, the code can be integrated to test environment to measure its behaviour in the production PDL.

9.3 Generalized Testing Targets

It is up to the developers to adopt the strategies to test a smart contract. However, the following generalized testing targets listed below shall be met.

[RGTT 1] Validate expected behaviour - verify the expected behaviour with the achieved one.

Improve code quality - the code shall be clearly and professionally designed.

[RGTT 2] Design efficient smart contracts by adopting efficient programming strategies (e.g. some built-in language functions perform better than others)

[RGTT 3] Using widely available libraries - that is, some libraries may be not generalized enough

[RGTT 4] Behaviour in Edge cases (e.g. Genesis block, divided-by-zero error, lack of input and memory including errors resulting from network failures).

[RGTT 5] Synchronization - check for sequential flow of the code. That is, for example, the pre-requisites shall be executed before the follow-up.

9.4 Testing Checklist

The desired output of a smart contract depends on its purpose. Therefore, testing checklist varies depending on the requirement and expectation from the smart contract.

Following conditions shall be checked explicitly and documented:

[RTC 1] Entry Functions are secured and does not create back doors to other functions without access-control - Which of the smart contract's functions allow entry from external entities. That is, the functions that are accessible to a caller of the smart contract.

[RTC 2] Termination Function - A smart contract shall have a safe and callable termination condition (see clause 6.4.1.5).

[RTC 3] Logic Functions - Functions that perform the operations of a smart contract shall be present. Without such functions a smart contract is not usable.

[RTC 4] Access Rights - Only authorized user(s) shall have access to the functions of a smart contract. At the testing phase, accessibility to different functions with different roles (e.g. admin and user) shall be verified and validated.

[RTC 5] Mandatory fields (clause 6.3.13)

This may include:

[OTC 1] Monitor inputs and outputs.

[OTC 2] Execution time and network latency effects.

[OTC 3] CPU and memory consumption.

[OTC 4] External dependencies (e.g. other smart contracts and external sources of information).

[OTC 5] Technological dependencies (e.g. software libraries).

9.5 Offline Testing

9.5.1 Introduction

In offline testing a smart contract is tested locally, without connecting to the production PDL.

[OOT 1] This may include a standalone test node or a group of nodes working as a testbed.

[OOT 2] Both the unit [RT 6] and integration tests [RT 7] can be done offline before the online testing, to validate the smart contract expected behaviour.

Some of the offline testing can be done through.

9.5.2 Sandbox Testing

Typically, sandboxes run on a single machine with several containers acting as nodes of that PDL. Some of parameters may not be accurate with sandbox testing such as transaction latency, which will obviously be very low when all the containers are in a same machine. However, sandbox testing is still helpful for measuring and validating the behaviour of certain aspects of a smart contract.

Sandboxes should emulate production environment with the exception of execution latency.

Following are some requirements of the sandboxes that shall be used for smart contract testing:

[RSBT 1] Use the same PDL type and version as the smart contract is expected to be installed to. For example, Hyperledger Fabric version 2.0 smart contract shall be tested on a Hyperledger Fabric version 2.0 sandbox.

[RSBT 2] The sandbox shall be using/downloaded from the same source as the PDL-type. For example, for Corda testbed, the ledger artifacts shall be downloaded/extracted from the verified Corda source.

[RSBT 3] Number of Nodes - If resources are available (e.g. enough computation availability), a sandbox shall use same/or close to same number of nodes as the production PDL.

[RSBT 4] Operating system of the underlying sandbox shall be same (or as close as possible) to the production environment (e.g. Kubernetes and Docker).

9.5.3 Testbeds

If the resources allow, it is always a good idea to use a group of test nodes and mimic a production PDL. Smart contracts tested on such test-PDLs can be tested for additional parameters such as transaction throughput.

Following are the requirements for testbeds:

[RTB 1] All nodes shall use same PDL-type and version number as the production PDL.

[RTB 2] All the smart contracts shall be using the same programming language, libraries, and software as they are expected to be using in the production environment.

[RTB 3] All the PDL nodes shall be using same software configuration as the production-PDL (e.g. operating system and developers' environment).

9.6 Online Monitoring

9.6.1 Introduction

In online monitoring a smart contract is installed on a production PDL and the outputs and its interaction with other entities (e.g. other smart contracts and foreign PDLs) are monitored.

9.6.2 Time-Limited Test

[RTLT 1] Before the online monitoring initiates, a smart contract shall be installed on the production PDL and be tested for a limited time.

[RTLT 2] All the participants of the PDL, shall be made aware of the test-status of the contract. That is, any transactions done by the test smart contract shall not be valid until the test time is elapsed.

[OTLT 1] After the testing time, the governance may choose to extend the lifetime of the contract and take it to production or terminate it for improvements.

9.6.3 Monitoring

[RM 1] A smart contract shall be monitored for its lifetime.

[RM 2] It is essential to monitor a smart contract continuously because despite of thorough testing, some of the cases may lead to unexpected (possibly harmful) outcomes.

This advantageous for both the improvement and debugging purposes.

9.6.4 Online Reports

[OOR 1] A smart contract can have a programmed function which can generate periodic reports to the governance.

[OOR 2] The parameters of the reports may be specific to the purpose of the smart contract and the discretion of the PDL governance.

[OOR 3] As such an approach will occupy bandwidth, it may be feasible to program reporting transactions to off-peak times.

These reports may include:

[OOR 4] Number of execution requests in a unit time.

[OOR 5] The input and outputs.

9.6.5 Decisions Based on the Reports

Following decisions can be taken based on the reports:

[OORD 1] Suspension and upgradation of smart contract.

[OORD 2] Adjusting/updating access control to smart contracts.

10 Updating a Smart Contract

10.1 Introduction

It is sometimes required to update a smart contract, for example, change its end date or owner details. Yet, smart contracts are immutable, so an installed smart contract cannot be amended. However, this can be upgraded through installing a newer version of the smart contract and deactivating the old contract.

10.2 Update Situations

There can be several situations in which smart contract updates may be required.

It is up to the governance, owners and stakeholders to make the decision when updates are required. Some of the scenarios are outlined below:

1) New terms are identified which need to be included in future versions.
2) Vulnerability found in the old contract.
3) Old contract reached its end date and stakeholders wish to continue using the contract (in this case, stakeholders may choose to change the end date of the old contract instead of installing a new version).

Following are the situations when a smart contract shall be updated, and the old version will be considered as invalid or obsolete:

[RSCU 1] New vulnerability identified in the programming language, or any library used in the code of a smart contract.

[RSCU 2] Change/update in governing laws or standards.

[RSCU 3] The smart contract is not acting as planned.

10.3 Strategies of Updating

10.3.1 Old Version

[CSTU 1] **Deactivating the old contract -** the old contract shall be deactivated properly which means, ensuring the end date is the past date and all the variables are deactivated. Note that, two versions of a smart contract shall not be operational at a same time.

[CSTU 2] **Back up data and variables-** the old version stays on the ledger even after the deactivation. However, some stakeholders may prefer to keep a local copy due to unforeseen circumstances such as deletion of the complete chain.

10.3.2 Technological Upgrades

Below are the requirements for the update of the smart contract updates:

[RSTTU 1] **Technological updates -** new versions shall follow the same technology (e.g. programming language) as the old version to avoid interoperability issues. However, in some cases it may be required some updates but the developers shall ensure the interoperability with other contracts and the PDL technology.

[RSTTU 2] **Vulnerability -** if a vulnerability is found in old technology used, the reason/cause of the vulnerability shall be fixed by using the newer version. For example, if a library used in the old version has errors, that library shall not be used in the newer version and an alternate version of the library or alternate library shall be adopted for future contract versions.

10.3.3 Upgrading Through Versioning

When a smart contract is updated, it shall follow the version number in continuation with the old contract. For example, if the old version was xx:xxx:01 the next version shall be xx:xxx:02.

10.3.4 Updating Steps

Once the governance and stakeholders agree to update a smart contract. All of the following steps shall be taken:

[RUS 1] Identify the changes to be made (i.e. functions or software library).

[RUS 2] Make the changes.

[RUS 3] Test the smart contract.

[RUS 4] Redeploy the smart contract - a new version of a smart contract shall not be deployed before it has passed the testing.

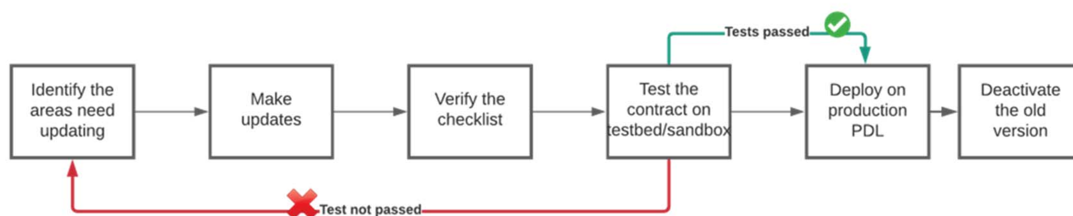

Figure 10-1: Smart contract update processes

10.3.5 Checklist Before Redeployment

These steps are mandatory and similar to initial deployment of the smart contract:

[RCKL 1] Passed the code test (test engineers or developers shall check and verify this) as per the guidelines listed in clause 9.

[RCKL 2] Passed the testbed/sandbox test (test engineers or developers shall check and verify this) as per the guidelines in clause 9.

[RCKL 3] If the test is carried out by test engineers, then confirmation is required from developers that the tests met the requirements. The testing requirements shall follow the guidelines listed in clause 9.

10.3.6 Securely Inactivating Old Contract

[RSIOC 1] Old version of the smart contract shall be terminated before or concurrently with the deployment of a new/updated version.

[OSIOC 1] The sequence is decided on a case-by-case basis by the stakeholders and the governance.

[RSIOC 2] In either case, the agreement or the smart contract shall not be affected by this transition from the old to the new/revised contract.

The requirements are as follows:

[RSIOC 3] The start date of the new version shall be after the termination date of the old version.

[OSIOC 2] The time between the start of new version and end of old version is up to the stakeholders and the governance but.

[RSIOC 4] In no circumstances two versions of a smart contracts shall be active at the same time.

11 Threats and Security

11.1 Introduction

To use smart contracts for contractual purposes, they should be secured and impenetrable. Threats and dangers to a smart contract are highlighted in ETSI GR PDL 004 [i.1].

11.2 Threats

11.2.1 Smart Contract Programming Errors

As discussed in earlier sections, smart contracts are immutable, therefore any error done in programming would make a smart contract perform invalid/erroneous executions. Such an action may cause people hefty damages such as monetary losses and reputational damages. It is the contract owners' and developers' shared responsibility that the testing checklist (clause 8.5) is followed before the deployment of the contract.

[RPE 1] Governance shall ensure that all the owners of the contracts are following the correct procedures for the coding and testing as specified in the present document and ETSI GR PDL 004 [i.1].

[RPE 2] A smart contract shall be tested against a pre-defined list of tests based on the requirements defined in clause 9 should pass those tests as a pre-condition to deployment.

11.2.2 Internal Threats

11.2.2.1 Transactions Ordering

Smart contracts may require information resulting from previous transactions.

[RTO 1] The governance shall sequence the transaction executions such that dependent smart contracts operate in a consecutive manner.

[RTO 2] A smart contract shall receive a correct data for its execution. Governance of the PDL shall keep track of PDL latency.

[OTP 1] May introduce a wait time before the execution of smart contract. For example, if a PDL has 5 ms transaction latency, the smart contracts will wait for this time before starting the execution.

11.2.2.2 Malicious/Accidental Executions

Albeit the access control mechanisms it is still possible that some of the users send incorrect data to the PDL. Such a behaviour can be benign or intentional.

[RME 1] Governance shall ensure that all the nodes (i.e. PDL participants) shall follow security protocols (i.e. SSL) to access the PDL to avoid attacks such as the man-in-the-middle attack.

[RME 2] Governance shall ensure that the nodes (i.e. PDL participants), shall not send invalid/wrong transactions.

[RME 3] Governance of the PDL shall introduce compliance strategies such as applying penalties (such as temporary blacklisting) to nodes that do not follow the protocol.

[RME 4] Node owners should ensure that they have adequate network resources to meet smart contract execution requirements.

11.2.2.3 Reporting Wrong Parameters

Users may advertently and inadvertently report incorrect data, which may affect the smart contract executions and results. For example, when a user is reporting its own device managed data (e.g. QoS parameters) and it is in their benefit to overstate their parameters. It is likely that they may send wrong/incorrect parameters to the ledger. In some of the cases when data is at very fast speed such as routers' data, due to speed of execution, it may be difficult to identify such behaviour in real time. It is not always the users who will try to send the wrong data to the ledger, other factors such as man-in-the-middle or benign mistakes can also result in wrong data inputs.

This can be mitigated through online monitoring (clause 9.6). The smart contract and associated data can be monitored, and if wrong inputs are deducted the smart contract can be terminated immediately.

If wrong parameters are reported following measures shall be taken:

[RWP 1] Node Owners shall take all the necessary measures that accurate and timely parameters are passed to a smart contract.

[RWP 2] Governance shall observe execution activities periodically and take necessary compliance measures of potential problems identified.

[OWP 1] Governance observation intervals/periods is dependent on case-to-case bases and up to the discretion of the governance.

Another solution may be to adopt interrogation protocol. In this protocol, the devices keep the local record of the data forwarded by them and only the details of the flow at the source and the destination is recorded in the ledger.

11.2.3 External Threats

11.2.3.1 Introduction

External threats are the dangers and threats to a smart contract from external entities such as foreign PDLs or oracles.

The governance shall ensure the safety and security of smart contracts and shall not allow the external entities to access the contracts without rigorous checking.

However, even the allowed calls if not checked for latency can cause the denial-of-service attacks. That is, a smart contract gets more executions than it can handle.

To avoid external threats following are the requirements:

[RET 1] Too many transactions - may not be able to process by a smart, may cause congestion at the ledger. Governance will be responsible to keep track of this.

[RET 2] Authorized access - the keys should be revoked without ANY delay.

[RET 3] Governance shall ensure the smart contract only gets as many calls it can handle.

11.2.3.2 Malicious Oracles

Oracles can be both internal or external (clause 7.5). Oracles can be malicious and send the wrong/delayed information to the PDL. In the earlier clauses, it is required that governance of the PDL shall maintain a list of trusted oracles:

[RMO 1] To ensure the timely data, Governance of the PDL shall define the threshold time to accept the data from oracles.

[CMO 1] This time would differ with the use case.

For example, for weather data, hourly update would be appropriate but for stock exchange a finer interval would be required.

[CMO 2] Oracles may be vulnerable to attacks such as bribery.

[CMO 3] Malicious Oracles cause denial of service attacks because if there is no API in the middle it can overwhelm the PDL.

11.2.3.3 Accidental Damages

[RAD 1] System should be able handle accidental attacks and minimize such problems.

11.2.3.4 Malicious Attacks

Some of the checks listed below can be adopted to achieve this:

[RMA 1] System should be self-protecting, that is a smart contract shall have mechanisms to pick up erroneous and malicious calls and shall mechanism to report such a behaviour to the governance.

[RMA 2] Malicious attacks should be penalized by the governance. Governance can maintain a list of entities/parties trying to behave maliciously and take necessary measures to block their future access.

11.2.3.5 Denial of Service Attack

[RMDOS 1] Governance of the PDL shall ensure all the external inputs to the PDL are through an API, that is, shall be checked before allowed in the PDL network.

[RMDOS 2] Number of transaction requests altogether (local requests and external requests) shall not exceed the throughput of the PDL.

11.2.3.6 Re-entrancy Attack

[REEA 1] To prevent this attack, the developers shall ensure that a smart contract is secure and impenetrable. Further details on ensuring secure smart contract development; see clause 6.3.

11.2.3.7 Numerical/Integer Overflow attack

This attack can be prevented through careful planning and thorough testing. Developers shall follow the guidelines in clause 9 for testing.

History

Document history		
V1.1.1	December 2021	Publication