ETSI GR PDL 004 - v0.0.14 (2020-10)

**Group Report**

Smart Contracts in Permissioned Distributed Ledgers System - Architecture and Functional Specification

***Disclaimer***

The present document has been produced and approved by the <long ISGname> (<short ISGname>) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference

DGR/PDL-004

Keywords

blockchain, Smart Contracts, Policy, Service Level Agreement (SLA)

***ETSI***

650 Route des Lucioles

F-06921 Sophia Antipolis Cedex - FRANCE

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

Siret N° 348 623 562 00017 - NAF 742 C

Association à but non lucratif enregistrée à la

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

***Important notice***

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

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

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

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

***Copyright Notification***

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

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

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

© ETSI 2020.

All rights reserved.

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

Contents

Intellectual Property Rights 6

Foreword 6

Modal verbs terminology 6

Executive summary 6

Introduction 7

1 Scope 7

2 References 7

2.1 Normative references 7

2.2 Informative reference 7

3 Definition of terms, symbols and abbreviations 7

3.1 Definitions 7

3.1.1 Smart contract 7

3.1.2 On-chain Smart contract 7

3.1.3 Off-chain Smart contract 8

3.1.4 Master-chain 8

3.1.5 Side-chain 8

3.1.6 Ricardian Contract 8

3.1.7 The Difference between Ricardian contract and Smart contract 8

3.1.8 Eternal Contracts: 8

3.1.9 Coin 8

3.1.10 Token 8

3.1.11 Stakeholders 9

3.1.12 Participants 9

3.2 Symbols: 9

3.3 Abbreviations 9

4 Introduction to Smart Contracts 10

4.1 Introduction 10

4.2 Smart Contract Programming Paradigms 10

4.2.1 Object-Oriented Paradigm 10

4.2.2 Properties 11

4.2.2.1 Immutability 11

4.2.2.2 Availability 11

4.2.2.3 Transparency 11

4.2.2.4 Self-Execution 11

4.2.2.5 Reusability 11

4.3 Storage 12

4.4 Life cycle of a Smart Contract 12

5 Smart Contracts – Lifecycle phases 12

5.1 Introduction 12

5.1.1 Planning Phase 13

5.1.1.1 Governance 13

5.1.2 Design Planing – Coding and Testing 14

5.1.3 Deployment Planning 15

5.1.3.1 Introduction 15

5.1.3.2 On-chain deployment 16

5.1.3.3 Side-Chain deployment 16

5.1.3.4 Off-Chain deployment 16

5.1.3.5 Immutable deployment 17

5.1.3.6 Terminatable deployment 17

5.1.3.7 Upgradeable deployment 17

5.1.4 Draft Template 17

5.1.4.1 Introduction 17

5.1.4.2 Terms Negotiation: 17

5.1.4.3 Map Draft template to the machine-readable context (Compile Draft): 18

5.1.4.4 Draft Review (Reference Checklist): 18

5.1.5 Coding and Testing phase 18

5.1.5.1 Introduction: 18

5.1.5.2 Coding process: 19

5.1.5.3 Testing process: 19

5.1.5.4 Code/Programming language level Testing 19

5.1.5.5 Smart Contract specific Testing 19

5.1.5.5.1 Open Source SC Analysers 20

5.1.5.5.2 Sandbox Testing 20

5.1.5.5.2.1 Three passes: 20

5.1.5.5.2.1.1 Introduction 20

5.1.5.6 Validation 21

5.1.5.7 User Experience Testing: 21

5.1.5.8 Consumer protection: 21

5.1.6 Deployment and Execution Phase 21

5.1.6.1 Deployment 21

5.1.6.2 Execution 21

5.1.6.3 Termination 22

6 Architectural requirements for Smart Contracts 22

6.1 Introduction 22

6.2 Architectural requirements 22

6.2.1 Reusability 22

6.2.2 Self-Destruction 22

6.2.3 Data ownership 22

6.3 Reference Architecture 22

6.3.1 Introduction: 22

6.3.1.1 Data retrieval in smart contracts: 23

6.3.1.2 Transactions and Transaction Dependencies 23

6.3.1.3 Smart contract architecture - Without Smart contract chaining 23

6.3.2 Smart contract architecture - with contracts chaining 24

7 Smart Contracts – Applications, solutions and Needs 24

7.1 Introduction 24

7.2 Applications 25

7.2.1 Introduction: 25

7.2.2 ICT Sector 25

7.2.3 Automated Machines/sensors 25

7.2.4 Automated Auctions/Sales 25

7.2.5 Mechanism for Access Control/Certification Authority 26

7.3 Solutions 26

7.3.1 Introduction 26

7.3.2 Scalablity 27

7.3.2.1 Check-point 27

7.3.2.2 Extensibility: 27

7.3.3 Security of contracts: 27

7.3.4 Example: Smart contracts with QoS monitoring 27

7.4 Needs - Requirements to build a viable system with Smart contracts 28

7.4.1 Regulatory Aspects 28

7.4.2 Security of the Contracts 29

7.4.2.1 Secure data feed(oracles): 29

7.4.3 Enforceability 29

7.4.4 Availablity 29

7.4.5 Attacks 30

7.4.5.1 Reentrancy 30

7.4.5.2 Free Option Problem 30

7.4.5.3 Denial of Capacity Attack: 30

8 Limitations of Smart Contracts 30

8.1 Inter and Intra system threats 30

8.1.1 Introduction: 30

8.1.2 Absence of Termination clause/Self-Destruction 31

8.1.3 Admission Control 31

8.1.4 Off-chain and side-chain contracts handling 31

8.1.5 Poor Exception Handling 31

8.1.6 Transparency of a PDL 31

8.1.7 External Libraries 32

8.2 Limitations 32

8.2.1 Introduction 32

8.2.2 Occupancy: 32

8.2.3 Latency 32

8.2.4 Underlying and Relying ledgers in permissioned context: 32

8.2.5 Not every term can be translated to a Smart Contract 33

8.2.6 Legal Uncretainity: 33

8.2.7 Intellectual Property Rights: 33

8.2.8 Accountability in Smart Contracts: 33

Annex A: Title of annex 35

Annex B: Title of annex 36

B.1 First clause of the annex 36

B.1.1 First subdivided clause of the annex 36

Annex C: Bibliography 37

Annex : Change History 38

Document History 39

#

# Intellectual Property Rights

Essential patents

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

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

Trademarks

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

# Foreword

*[ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx)* [(](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx)*[EDRs)](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx),*

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

The present document is part <i> of a multi-part deliverable. Full details of the entire series can be found in part [x] [Bookmark reference].

# Modal verbs terminology

*[ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx)* [(](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx)*[EDRs)](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx),*

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the [ETSI Drafting Rules](https://portal.etsi.org/Services/editHelp%21/Howtostart/ETSIDraftingRules.aspx) (Verbal forms for the expression of provisions).

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

# Executive summary

This present document specifies a high-level functional abstraction of PDL Smart Contract System Architecture. In particular, basic building blocks for designing, coding and testing Smart Contracts for the PDLs. This includes describing how different classes of systems interact with Smart Contracts. Processes, models, and detailed information are beyond the scope of the present document.

# Introduction

The present document defines a high-level functional abstraction of policies to design and code Smart Contract components. Smart Contracts are mere codes, and if not well planned, designed, coded and tested; can leave the system vulnerable to external attacks and internal errors.

# Scope

**Scope of work to be undertaken:** The present document specifies the functional components of Smart Contracts, their planning, coding and testing. This includes:

a) reference architecture of the technology enabling Smart Contracts – the planning, designing and programming frameworks

b) specify how to engage using this architecture – the methods and frameworks the Smart Contracts building blocks possibly communicate

c) point out possible threats and limitations

# References

## Normative references

## Informative reference

#  Definition of terms, symbols and abbreviations

## Definitions

### Smart contract

As per ISO /TR23455/2019 definition, Smart Contract is:

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

***A smart contract might represent terms in a contract in law and create a legally enforceable obligation under the legislation of an applicable jurisdiction.***

**A smart contract may but doesn’t have to be human readable and must be self-execuatable.**

Any executable code stored on a PDL is dubbed a “Smart Contract” (SC).

In this document we separate smart contract and smart legal contract.

### On-chain Smart contract

The on-chain smart contract is the contract that resides in the master-chain and is executed directly without the instantiation of any other contract. The beneficiaries get rewarded immediately, as soon as the contract is executed without the involvement of any other contract.

### Off-chain Smart contract

Off-chain contracts are the smart contracts stored away from the ledger(i.e. trusted database or side-chain) and their execution may depend on on-chain contracts(i.e. on-chain contract can initiate off-chain contracts) and later the state can be updated.

### Master-chain

The primary chain where the executions of the smart contract are recorded.

### Side-chain

The chain(s) which work as a secondary chain to main chain/ledger. It can be used to off-load some of the computations for the purpose of scalability or privacy.

### Ricardian Contract

A Ricardian contract is a single contract document which is both easily readable by human and machines and not self-executable. It is formatted as a text file and digitally signed by the issuer of the contract.

The security of a Ricardian contract is achieved by OpenPGP and all the signing keys are included within the contract so eliminates the use of external certificate authority or we can say that a Ricardian contract carries its own PKI with them.

### The Difference between Ricardian contract and Smart contract

The major difference between the smart contract and the Ricardian contracts is that Smart Contracts are executable code but Ricardian contracts are the agreements recorded in a single file and not executable on their own. A smart contract doesn't have to be a Ricardian contract and a Ricardian contract is not a Smart contract, but a Smart contract can execute a Ricardian contract.

Table 1: Comparison of Ricardian and Smart Contract

|  |  |  |  |
| --- | --- | --- | --- |
| **Contract Type** | **Machine-Readable** | **Human-Readable**  | **Self-Executable** |
| Ricardian Contract | Yes | Yes | No |
| Smart Contract | Yes | Optional | Yes |

### Eternal Contracts:

The contracts which are active for infinite time.

### Coin

A coin is implemented using a unique ledger and is usually used for financial transactions (e.g. Ether, Bitcoin).

### Token

A token can be implemented using a any ledger and is usually used to represent assets or ownership of assets or rights to perform actions. For example the right to participate in an auction or representation of an item being sold.

### Stakeholders

Stakeholders are the parties that benefit from the PDL. All the stakeholders may or may not keep the copy of the ledger (i.e. act as node) and take part in consensus.

### Participants

Participants of the PDL keep the copy of the ledger and take part in the consensus.

## Symbols:

##  Abbreviations

AML: Anti-Money Laundering

API: Application Programming Interface

SC: Smart Contract

CEN-CENELEC: European Committee for Standardization and European Committee for Electrotechnical Standardization.

DLT. Distributed Ledger Technology

EBP: European Blockchain Partnership

EBSI: European Blockchain Service Infrastructure.

EC: European Commission

EFTA: European Free Trade Association

eIDAS: Electronic Identification, Authentication and Trust Services.

EIRA: European Interoperability Reference Architecture

ESSIF: European Self Sovereign Identity Framework

ETSI: European Telecommunication Standards Institute

EU. European Union

FIG: International Federation of Surveyors

GDPR: General Data Protection Regulation

HPLMN: Home Public Land Mobile Network

ICO: Initial Coin Offering

ICT: Information and Communications Technology

ISO: International Standards Organization

ITU: International Telecommunication Unit

KYC: Know Your Customer

OECD: Organization for Economic Co-operation and Development

PDL: Permissioned Distributed Ledger.

RA: Regulatory Authority

SG: Study Group.

SLA: Service Level Agreement

SME: Small and Medium Enterprise.

STO: Security Token Offering.

TOOP: The Once-only Principle

TSAG: Telecommunication Standardization Advisory Group.

UN/CEFACT: United Nations Centre for Trade Facilitation and Electronic Business.

UNCITRAL: United Nations Commission on International Trade Law.

UNE: Spanish Association for Standardization.

VPLMN: Visiting Public Land Mobile Network

QoS: Quality of Service

# Introduction to Smart Contracts

## Introduction

In general, an computer program deployed on a PDL can be considered as a SC, but different schemes are possible to achieve this. In the next two sections, we will show the two main proposals adopted in practice, highlighting their similarities to classical programming languages paradigms.

## Smart Contract Programming Paradigms

Any PDL’s general goal is the distributed management of a common data repository defining a current global state; there is no assumption on the type of data stored. When such data is an executable code, the induced global state can be seen as the state of a distributed virtual machine.

### Object-Oriented Paradigm

Historically, the main model adopted for SCs has been on the line of the traditional Object Oriented paradigm. As such, a SC is seen as a code entity composed of two main sections:

* Internal storage, in the form of identifiers – value associations akin to a dictionary, similarly to object fields;
* Functions’ definitions, specifies the set of actions allowed for the given SC with the appropriate scope modifiers, similarly to object methods.

Similar to the concepts of Object oriented programming a smart contract is instantiated from a class and once instantiated holds an unique identifier; that is to say every instantiation is unique. The deployed smart contract holds a global state which means that all its fields and functions become visible and callable by other contracts (depending on access rights). Moreover, a deployed smart contract can be called as many times as required, however this is dependent on the implementation.

Protocols for smart contracts have different implementations depending on the technology and consensus mechanism such as PDL types (e.g. Ethereum, Quorum etc.)

### Properties

Smart contracts’ properties directly depend on the properties of the underlying PDL and some properties due to their requirements .

#### Immutability

As any data on a PDL, a SC is immutable; this means that a smart contract code, once accepted through consensus, cannot be changed. However modifications through other methods such as proxy contracts or introducing a new smart contract, are possible. In such event, the old version of the contract exists in the chain. As consequence of immutablity is Immortability which means that it cannot be deleted from the ledger after deployment. This brings the challenges of scalability as a PDL might be populated with dormant contracts over the time. The details on scalability is discussed in later sections.

The values contained inside a SC’s internal storage are mutable as expected through function calls for example in an auction contract bid values will change with new bids but the final winning bid may be immutable.

#### Availability

In the case of on-chain SC is always available as long as the underlying master ledger is accessible. This means that a SC function can be invoked, and its fields (i.e. variables) can be read, by an entity as long as the entity has the appropriate privileges specified by the contract and the PDL. However, in the case of off-chain smart contracts, if the ledger where the contract is installed (i.e. secondary PDL) is not available, the smart contract is not accessbile by the master PDL.

#### Transparency

Any entity, with the appropriate privileges, might inspect a SC code and current values. As such, it is transparent to all intended participants of the PDL. Transparency should not be confused with immutablity; a contract code remains unchangeable even though it is transparent to both parties.

Moreover, any call to a function of a contract is performed through a general state update on the PDL (i.e. transaction). As such, all function calls are recorded in the PDL and traceable by the members of the PDL with appropriate access rights.

#### Self-Execution

Any execution of a SC, i.e. an invocation to one of its visible functions, is performed by the PDL nodes, not by the user invoking the SC, nor by the SC creator. The SC execution is protected by the distributed consensus of the PDL, as such, it is beyond the control of any single party to execute a smart contract without the approval of PDL members. This property induces the sub-properties of:

* *Atomicity*: a SC invocation runs entirely or fails without affecting the state(i.e. there is no such thing as partial SC execution);
* *Synchronicity*: a SC invocation is executed in a synchronous way(i.e. every member with appropriate access rights get the update);
* *Determinism*: a SC invocation returns the same result for any node executing it.

#### Reusability

SCs are coded once and can be executed multiple times dependent on the PDL governance. A given smart contract can be used as a template for a wider set of applications sharing the same high level logic. The actual behaviour of a given contract may change depending on the parameters which are set at invocation time. For example, the SC for cellular service is modelled with required fields for QoS metrics such as latency; all the telecom operators, in this case, will be required to specify the latency as a parameter.

## Storage

Smart contracts are stored in distributed ledgers; however, their storage depends upon the nature of ledger architecture. For example, in case of a public blockchain such as Ethereum, a smart contract will be stored by all nodes; on the contrary, in a permissioned blockchain such as Hyperledger, smart contracts are stored only on the nodes that are part of a given channel (an abstract point-to-point link between nodes) and are established through communication between nodes.

Resuability techniques such as template contracts can be used to allow efficient storage of the contracts. The decision of storage is dependent on the implementation of a PDL, and the technology companies adopt. The limitations due to enternal exisitance of the contract is discussed in Section 6.

## Life cycle of a Smart Contract

A smart contract is a computer program, the difference is that the smart contracts are immutable, so it requires great care to program them and should be tested on several levels before deployment. This section presents the recommended life cycle, a smart contract should follow in order to avoid the dangers such as errornous code. This recommended lifecycle consists of three phases: planning phase, coding & testing phase and deployment & execution phase. The phases are explained in detail in Section 5.



Figure 4‑1: Life-cycle of a smart contract

#  Smart Contracts – Lifecycle phases

## Introduction

Smart Contracts are software codes, and similar efforts are required to program them like any other software program. The difference is, however for any usual software, the bugs can be fixed in new releases or through software patches, but SCs stored in a PDL by design lack this provision. The deployed SC code becomes ossified to the system and further changes to the same code under the same identification is not permitted(i.e. version mechanism can be applied to installed an updated contract). Several researchers have identified instances, where a bug or loop-hole in the contract resulted in loss of tokens, for example, the DAO attack costed $60 million and Ethereum had to create a hard fork to overcome the problem.

Hence, the careful planning and scrutiny of the code before deployment to the ledger is of utmost importance. In this section, we explain the stages of the smart contract life cycle, which industry may follow to adopt the PDL.

Smart contracts are digital representation of the contracts and should not be confused with legal contracts. They may represent legal contracts but this section specifically focused on smart contracts as software codes.

### Planning Phase

A smart contract can be deployed in many ways and all the methods are dependent on its underlying ledger technology and should be acceptable by the participants through consensus. Our goal here is to create a contract that can be trusted by participants who do not trust each other. The planning of a smart contract will enable the participants define their requirements and functionalities of a smart contract. Planning phase may include: 1) Governance - ownership and access rights 2) design – coding and testing 3) Deployment and 4) Management planning.

#### Governance

A Smart contract may define a contract, and its associated terms and conditions covering the full life cycle of the contract, between the participants. Smart contract is a digital collection of rules representing and implementing the above in an electronic manner. Governance planning defines the authority of different stakeholders over the contract, for example, ownership and access rights.

Usually, the creator of a contract is the owner as well; the owner of the contracts have exclusive privileges such as contract destruction. However, in PDLs where contracts can be reused by several participants for several unrelated transactions, it is feasible to have a role-based ownership mechanism. In *Role-Based ownership*, the operations of a contract are governed by a group of participants with appropriate privileges; as PDL is a collaborative ledger, these privileges can be specific to a contract.

Single-party Governance:

The smart contract, when deployed, is usually identified as being *governed* by a specific part (N=1) or a group of distinct parties depending on consencus and governance model.

This agreement needs to take into account the legal and business aspects of the smart contract, and address issues such as who is eligible to stop, terminate, or upgrade the smart contract, and how these are enforced contractually or technically.

Smart contracts are digital model of such contracts and actors and their arrangements is beyond the scope of this document.

Multi-party Governance:

A smart contract may be developed for the purpose of N-M interaction, i.e. one-to-one, one-to-many, many-to-one or many-to-many interactions. For example, if the contract is governed by more than one party, a *consortium agreement* needs to be formulated within that group to outline the governance model that is applied to the smart contract. Moreover, a contract may be managed by a third-party such as some stakeholders which are not directly involved in the contract.

For multi-party governance this requires decisions on the technical implementation aspects of:

* ***From whom smart contract will accept the operational decisions and how?*** Since in this scenario, smart contract is governed by multiple stakeholders, it is likely that some of the authorised parties/stakeholders may disagree to some decisions such as termination of a contract. In such cases, multi-signatures and voting mechanisms can be used to approve/reject a transaction.
	+ In multi-signatures, group members sign a decision that is communicated to a smart contract and verified.
	+ Another option is to use voting, in which case an action is initiated, but the smart contract requires different parties to individually endorse the action (or reject it) within a time limit.
* ***How are the governing parties recognized by the smart contract?*** Depending on the ledger, this may be an organizational identity within the ledger, or an account owned by the party(e.g. a public key).
* ***What are rights each governing entity has?*** It is possible that some ledgers don’t allow some actions, such as contract stop and resume, termination, contract upgrade, changes in governing party identities, and any other business-specific actions.
* ***How the smart contracts will be upgraded?*** If the smart contract can be upgraded, either via the ledger’s native support (e.g. in Hyperledger Fabric, using versioned chaincode), or via development techniques (e.g. proxy contract), the process of upgrade needs to be managed. This may require communication with the users of the smart contract as with any software release management process. If the smart contract is governed by a group, they must coordinate the upgrade using the appropriate technical means.

### Design Planing – Coding and Testing

Smart contract must be designed with special care as there is no provision of amendment(i.e. smart contracts by definition are imutable). Design planning may include choice of computer languages used for the contract programming and processes are highlighted in Figure 2. The coding and testing is detailed in Section 5.1.4.



Figure 5‑1: Smart Contract Desgin Process

### Deployment Planning

#### Introduction

Smart contracts can be deployed on the master-chain, side-chain or off-chain depending on the planning and requirements of the organizations. For example, if two companies are willing run a business contract that should stay exclusively between them, they can have a side-chain with smart contracts deployed there and make appropriate selective updates to the main-chain such as contract start and termination dates without the details of the contract.

Table 2: Explaination of Master-Chain, Side-Chain and Off-Chain

|  |  |  |  |
| --- | --- | --- | --- |
|  | Master-Chain | Side-Chain | Off-Chain |
| Contract-type |  Contract,Address of Contracts on Side-Chain and/or Off-Chain | Contract | Contract |
| Scalability | Limited | Limited | High |
| Security | High | Limited(see section X for details) | Requires off-chain security measures |
| Immutability | High | Ledger-dependent | Limited |
| Eternity | High | Ledger-dependent | Limited |
| Risks | Low | Medium  | High |
| Storage-requirement | local | Can be distributed | Doesn’t need to be shared  |
| Speed | Medium | Slower  | Faster |
| Dependency | None | Ledger and governance dependent | Ledger and governance dependent |
| Parallelization | Ledger-dependent | Ledger-dependent | Governance dependent |

In the following paragraphs, we discuss possible methods of deployment.

#### On-chain deployment

This is the simplest method for deployment of smart contracts and the contracts are stored directly in the master-ledger. The advantage is that the customers do not have to rely on any other side-chain or off-chain (which may require additional resources) and it is best for a system manged by a single entity. Since all the full contract codes are stored in the master-chain, in long-term scalability can be a problem.

The simplest deployment model is where the smart contract is never terminated. In some ledgers, a smart contract can always be removed, while in other ledgers this decision can be built into the smart contract at development(i.e. self-destructable clauses) or deployment time (i.e. by choosing to include or omit a “termination” mechanism self-destructable clauses are discussed).

#### Side-Chain deployment

In this method, the main logic of a contract is stored in a side-chain and only some indication of that contract (such as hash or address) is stored in the master-chain. The advantage of this technique is that, since it is not required for a full-contract code to be in the master-chain, this technique is scalable.

Additionally, the side-chain contract address in the master-chain can be updated by the owner of the contract through a transaction with no additional means. The danger in this type of deployment is that, if the side-chain contract is not self-destructive, it can stay forever and can be callable by other contracts, also as it is in the chain (no matter if the chain is side-chain) it occupies storage.

Side-chain smart contracts can be reused by other users of the PDL (delegated by the owner of the contract).



Figure 5‑2 Master-chain and side-chain Smart contracts

#### Off-Chain deployment

In off-chain deployment smart contracts are stored away from the ledger and may be in a trusted data structure. The indication of the presence of contracts such as invocations are only recorded in the master-chain or a side-chain. Off-chain deployment possess risk of trust and rely on security of the database where the contracts are stored. The major advantage of an off-chain deployment is, this technique is scalable since only the invocations are stored in the PDL. Since, off-chain deployment doesn’t depend on any specific PDL, such contracts can be ported to other PDL types with relative simplicity.

#### Immutable deployment

There are methods by which smart contracts’ immutability can be managed. This is typically done at the deployment stage. Some of immutability management techniques may be available natively in a specific ledger, and for other ledgers, this may require the use of programming techniques such as call delegation across contracts. We discussed immutablity as a property in detail in Section 4.2.2.1.

If the ledger has immutable smart contracts, this governance model must also be encoded within the smart contract during the contract planning, as any changes later may be difficult to implement.

#### Terminatable deployment

A smart contract may be terminated, i.e. permanently removed from the ledger, if the ledger or the smart contract itself directly supports this mechanism. A PDL is typically immutable so that smart contracts, but some ledgers may allow the contracts to be terminated and is dependent on the governance and the consensus of the under-lying ledger. (TBD: should discuss multi-party contract somewhere in the document – possibly link with PDL 003)(possible methods voting or multi-sig)

#### Upgradeable deployment

Some ledger technologies support upgrades to an existing smart contract, i.e. changing the smart contract’s operational code. This typically happens by installing a master contract with mutable field similar to passing an argument to a function. This argument acts as a pointer to another contract which carries the actual operational code. This type of deployment is useful when upgrades of a contract are required. However, in this case the problem of scalability exists because the old contracts may not be deleted and stay in the ledger as a dormant contract.

If the smart contract can be upgraded, either via the ledger’s native support (such as in Hyperledger Fabric, using versioned chaincode), or via development techniques (proxy contract), the process of upgrades needs to be managed. This may require communication with the users of the smart contract as with any software release management process. If the smart contract is governed by a group, they must coordinate the upgrade using the appropriate technical means.

### Draft Template

#### Introduction

In future, the PDL technology is envisioned to be used widely for all kinds of business transactions. Therefore, before the planning and coding process begins, a smart contract should be drafted electronically or manually. At this initial stage, some or all of the stakeholders can decide together with their requirements such as code and resources requirements. This step facilitates the smooth and error-free coding of a contract.

#### Terms Negotiation:

Once the draft of requirements is ready, the terms and conditions between the stakeholders should be decided and agreed. It is particularly important in a smart contracts because in traditional manual contracts, there may be a freedom of amendment at any time, whereas smart contract by-design do not typically have such freedom. At the same time, it is important that all stakeholders agree on terms of the entire deliberation so that there is no conflict in the future.

The terms and conditions will be varied from organisation and its governance, but questions such as deployment management and life-cycle of a smart contracts can be addressed.

Some of the major points that could be a part of the terms negotation:

1. Is the smart contract going to on-chain or off-chain?
2. If participants want to maintain a side-chain, who will be particpants and their role?
3. For how long the side-chain will be active?

Especially in situations where contract can be stopped and resumed, terminated, or upgraded, the multi-party governance agreement must take into account who has the authority to issue these operations.

Depending on the capabilities of the ledger itself some of these policy decisions may be part of the ledger itself; in other cases, these decisions must be encoded into the smart contract and defined in design phase already.

#### Map Draft template to the machine-readable context (Compile Draft):

This step provides the bridge between the draft template and the coding phase, and involves the procedures to map the draft contract (from template draft Section 5.1.4) to a smart contract which is the technical representation of the same. This step should not be confused with “Compile” in the context of programming and only harmonises the template and coding steps.

This step, can specify the complete supervisory level specifications such as, underlying ledger technology to be used, the stakeholder needs etc.

#### Draft Review (Reference Checklist):

The last step of planning phase should be to review and verify the complete planning phase. The reference checklist may include:

1. All the stakeholder requirements are listed in the draft.
2. The planned hardware and software resources such as PDL are acceptable and reachable to all of the future nodes (i.e. participants).
3. All the functions are mapped accurately to the requirements.
4. The governance of the contract is clearly documented and agreed by the stakeholders and part of the draft.
5. The contract is planned in accordance with the standardisation body guidelines.

### Coding and Testing phase

#### Introduction:

As soon as the contract plan is in place, the next step is to code it. In this section, we will cover the coding and testing phase of a smart contract and discuss the steps which can help industries produce viabale contracts.

#### Coding process:

The coding language decision is dependent on the underlying PDL type. Some of the PDLs may allow different languages for the Smart contract coding, but some are very specific to this. Where there is a freedom provided by a PDL type to use multiple languages, the widely used language may be adopted as they are better understood by the programmers and may have more tools available for testing and bug fixing.

#### Testing process:

A smart contract should go through a comprehensive testing process, to avoid errornous contracts being deployed. There are several steps can be the part of this process and depends on the priority of organisations. A recommended testing flow is shown in Figure 3.



Figure 5‑3: Smart Contract Testing process

#### Code/Programming language level Testing

The Smart Contracts’ testing varies from traditional software testing in several ways.

Traditional software mostly has the freedom of revision. When needed, it goes through regular updates, software revisions and patches to remove bugs. Smart contracts are deployed on PDLs, this means all the nodes carry a copy of the same contract and execute as required. Also, any syntax or logical error will be replicated to all of the nodes and it is impossible to fix such errors once the contract is deployed in the PDL. Such errors can be avoided by traditional language specific software testing mechanisms. Programmers should ensure that a contract is error-free and carry out all necessary tests in a test envoirnment before deployment.

#### Smart Contract specific Testing

Smart contract testing is different from code level testing as this level of testing ensures a safe and manageable smart contract. Smart contracts are typically auto-executable and their termination is difficult, hence it is important to consider following while testing a smart contract.

##### Open Source SC Analysers

A number of open-source SC analyzers such as Securify[[1]](#footnote-2) and SmartCheck[[2]](#footnote-3) are available to analyse the SC code and tag the vulnerabilities present in the program. These vulnerabilities such as unintiated functions, can provide third-party (possibly malicious) access to a contract, thus to the ledger. These analyzers prevent external accesses by inspecting the code and flagging the possible vulnerablabilities in the code. However, all the analyzers have their limitations such as they support certain ledger technology or programming language, also the attacks on the contracts are evolving, hence more comprehensive scrutiny of the contracts can be achieved by multiple analysis techniques. Another important consideration for an analyzer is the support for a PDL type, most of the available analyzers are for Ethereum and Hyperledger and the adopters of the other ledger types should look for their respective PDL supported analyzer.

##### Sandbox Testing

A PDL is a group of nodes, and errorounous smart contracts can be harmful to all of the nodes. A Sandbox testing mechanism is required before execution of a smart contract on a in-production PDL. Sandboxes are specific to the ledger type and can be local or distributed.

*Local Sandbox:*

A local copy of ledger can be used as a sandbox and sample contracts should be run several times to verify the output. A disadvantage of local testing is that it may not give the realistic latencies for execution and deployment. A solution for this can be a distributed full-scale Test-net.

*Distributed Sandbox/Test-net:*

A solution for limitations of local sandbox can be a permenant a sandbox between the nodes or a Test-net, which serve as the testing ground only and all the smart contracts deployed there should not be considered as valid; to enable scability in such sandboxes, they can be deleted after a certain time to free storage.

###### Three passes:

Introduction

It is recommended that nodes run their pre-tests before sending the deployment transaction. These pre-tests are specific to the use-case and the PDL type. For example, in a token contract the address of the payee must be included in the contract and for the asset trail contract, the change of owner is an important parameter. Here, we highlight three reference passes for a contract, stakeholders should look for, before depolyment of their smart contract:

*Execution clauses:*

A contract is executed with certain predefined conditions which can be internal such as start time or external such as an API call. Hence, it is important to have clearly defined execution clauses in a contract, as its absence will make the contract dormant. Moreover, the presence of unintended conditions can open backdoors in a contract and should be avoided.

*Penetratable clauses:*

The clauses that invoke the critical parts of the contracts such as payment remittance should be accessed exclusively by the owner or the authorised member of the PDL. Moreover, all the entry points to the contract should be examined to ensure water-tight security. Hackers usually exploit such loopholes or openings to gain access the contracts.

*Termination caluses:*

Smart contracts by-definition cannot be destructed but become inactive.Termination clauses allow the contract to stop its execution and become inactive; this prevents the ledger from having eternal contracts. Moreover, after a specific time, a contract should be self-destructible to avoid outdated versions of the contracts and allow the modified new versions.

It should be noted here that smart contracts varies in certain ways from legal contracts which can’t self-destruct but may include clauses after which those contract become ineffective.

#### Validation

The smart contract should be the exact and true representation of the natural language contract and should perform only the tasks specified there. In other words, semantic gap between the expected and the actual execution must be eliminated to avoid the wrong doings of a contract and implement an error-free code.The sementic gaps should be checked at the Level 3 of testing process(Figure 3).

#### User Experience Testing:

A group of users should test a smart contract on a sandbox. Their feedback will help in two ways: 1) the future users of the product can comment on the quality of the contracts and future development, and 2) Identify the errors and semantic gap in the contracts.

#### Consumer protection:

It is recommended to exercise the disclosure of minimum terms and conditions to transfer libability from the developers to the user. User should take full responsibility for the protection of sensitive data such as keys as leakage of information can put other PDL members’ data at risk.

### Deployment and Execution Phase

#### Deployment

Smart contracts by-design once deployed cannot be changed or amended. Hence, extensive emphasis on careful planning and design has been placed on the earlier stages. In the deployment stage, the contract is installed on a PDL, and it particularly involves the stakeholders such as a mobile operator and a tractor vendor, who agreed on a contract for network services. This stage should not necessarily involve the developers as the deployment can be straightforward if the earlier steps are carried out correctly and the pre-tested template of a required contract is available.

#### Execution

Deployed contracts can be executed unlimited times (depends upon the under-lying PDL type) during the execution phase. The execution of a smart contract can be parameterized and non-parameterized depends on the design model and can be performed by any authorized party through an API. Rest APIs can be used here and the payload can be implementation dependent.

#### Termination

A Smart contracts must be terminated exclusively or they should be self-destructable after certain a time as they may contain critical conditions such as payouts. In this case, if a dormant contract exists in a ledger can be exploited by the adversary. The termination of the contract can be done by the contract itself(i.e. destroys itself) or through an API handled exclusively by the stakeholders through the digital-signature mechanism, to ensure security. The termination should exclusively be performed by the owner of the contract, and it is possible that instantiation of one contract terminates the older one.

#  Architectural requirements for Smart Contracts

## Introduction

Smart contract depends on the PDL type and their architecture is also dependent on the PDL support. However, smart contracts must be carefully architectured to make them a workable and adoptable solution for the industry application. In this section, we discuss the architecutural requirements for a viable smart contract.

## Architectural requirements

### Reusability

Since a smart contract is a software that can live forever in a PDL, its architecture should be able to provide flexibility for reusability that is to say a contract may be generalised enough to be used multiple times. In a smart contract key parameters such as start date, end date and a beneficiary information can be specified to allocate a smart contract to several users.

The reusability can prevent the dormant contracts and the PDL being populated, thus helps in scalability.

### Self-Destruction

As discussed in Section 5, Smart contract may be destroyed or terminated after some time to avoid dormant or eternal contracts. However, some contracts are not suitable to be destroyed or terminated completely. For example, contracts with some monetary value can't be terminated because their destruction will cause the customers to lose funds. However, if a contract is some kind of agreement, for example, an agreement between a user and their network service provider, it should include the self-destructive clause.

[More should be added …]

### Data ownership

A smart contract should comply with GDPR requirements and keep public data only. If a smart contract wants to access or keep the private data (i.e. under certain licensing restrictions), the governance of the PDL may take and record appropriate permissions from the owner of the data.

## Reference Architecture

### Introduction:

A smart contract development may go through three different processes, 1) Logic – in which the original purpose of a smart contract is defined, 2) Algorithm – the code logic and the interpretation of logic to execution and 3) Code - the final code which must be a true representation of the initial planned Logic. In Figure 5, these processes are illustrated.



Figure 6‑1: Processes of a Smart contract

#### Data retrieval in smart contracts:

A smart contract may retrieve data from external sources such as oracles through an API. This access should be in compliance with the governance of the PDL and the country laws such as GDPR.

As discussed in Section 6.2.3, the data added to this smart contract should comply with GDPR – smart contract should keep only public data - private data should comply with regulations.

#### Transactions and Transaction Dependencies

A smart contract deployment and execution creates a transaction which is recorded in the PDL.

It is most probable that a smart contract is dependent on other contracts, in this case, the order of execution of these contracts is of utmost importance. The contracts that are dependent on the transactions from other contracts cannot be executed in advance and must wait for their pre-requisites to be completed.

The transaction ordering for a smart contract must be defined in the consensus of the corresponding PDL. It is recommended to adopt specific ordering of transaction inside the base contract (i.e the contract which will initiate the chain of contracts) to avoid transactions being rejected and cause clutter in the ledger because even the rejected transaction is recorded. With specific ordering all the dependencies must be clearly stated or referenced in the base contract that will access and all the transactions will follow the specified ordering(add appropriate time delay).

#### Smart contract architecture - Without Smart contract chaining

Following reference architecture defines a smart contract with recommended processes. Organisations may choose their own variations depending on PDL type and requirements of the use-case and may consider this as a initial design.



Figure 6‑2: Reference Architecture of a Smart Contract without contract chaining.

### Smart contract architecture - with contracts chaining



Figure 6‑3 Reference Architecuture of a Smart Contract with contract chaining

#  Smart Contracts – Applications, solutions and Needs

## Introduction

Smart contracts and their properties can be useful in many applications. Smart contracts can be applied in any DLT scenario where an automated and transparent contractual mechanism is required. However, the limitations and implications of adopting a smart contract based solution should be considered. In this sections, some of the possible applications and limitations of smart contracts are highlighted along with potential solutions.

## Applications

### Introduction:

Smart contracts can potentially be a viable solution for applications where transparency and immutability are a priority. They provide a mechanism to automate the contractual process, track the contract executions, and provide accountability in the contractual process. There are several ways and solutions where smart contracts can be applied to achieve the goals mentioned above, and some of them are highlighted here:

### ICT Sector

In the ICT sector, there are number of ways a digital service provider and a customer, no- matter business or an individual, engage in contracts. For example, Home mobile provider and Visited mobile provider have contracts for roaming services; the services consumed by the customer in the visited location is recorded and sent by the visited provider to the home provider. Smart contracts can automate this procedure by enabling service providers to create smart contracts for such digital services; as soon as the visiting customer consumes the network services of the visited operator, the corresponding smart contract is activated and enables instant settlement between the host and the visited provider including availability of the credit and payments.

Furthermore, mobile operators may not offer same performance in every time and space; factors such as congestion in the area and day/time impact the performance[reference]. This may result in a violation of the SLA between the user and the service provider. In situations where the mobile operators can’t provide the required QoS, possibly due to the congestion, customers may consider getting the services from other operators who offer a service guarantee. These provisions require automaticity and transparency. The customer wants to get the services instantly and automaticity. In the scenarios where QoS is of paramount importance(e.g. services for life-relying activities such as remote surgery), strict SLAs must be honored, and if the violation happens, the customer is notified(Transparency) and potentially compensated. Smart contracts can help to achieve these targets and provide a contractual framework in an un-trusted environment. This is achieved through logging of SLA and performance data on a PDL, and applying a smart contract to calculate the actual performance against the targets and automatically calculate the penalties according to the SLA where applicable, penalties can be automatically reduced from the invoice on the next billing cycle.

### Automated Machines/sensors

Automated machinery such as tractors and solar farms are equipped with sensors; these sensors transmit the device data such as engine readings or battery life to the Cloud or command center, where this information is processed to make future decisions such as capacity planning. Such systems are vulnerable to eavesdropping, replication, and man-in-the-middle attack. The attacker can pretend to be a legitimate device and send erroneous or incorrect data to the command center, and the valid user can be blamed for sending false/fake information. Such attacks can be mitigated using smart contracts; which can be installed on the ledger and while transmitting the sensor data, the unique identifier of the sensor sent along with the data, this information will be recorded as part of smart contract execution, which can verify the identity of the sensor. It is expected that data is sent with in a quantum-safe encrypted form to mitigate man-in-the-middle attack and eavesdropping.

### Automated Auctions/Sales

Automated auctions are found in almost every field. For example, telecom regulators auction bandwidths to operators. Smart contracts can help automate this process in such a way that the bandwidth contract is installed on a PDL with pre-defined parameters. An auction starts and ends with pre-defined time and all the bids are recorded in a PDL. This process becomes transparent to all the parties preventing dishonesty both by the bidder and the auctioneers. These bids can be tailored for specific needs for visibility and automated actioning.

### Mechanism for Access Control/Certification Authority

Smart contracts may be used as a mechanism for access control; as by definition they execute automatically, all the access information (e.g. user credentials) can be recorded in a PDL. For Example, a smart contract can be executed when some access rights are granted by a PDL-based certification authority. This may prevent the future disputes of data breach and provide a record of all the information exchange and key distribution.

In another example, Certificate Authorities are trusted by the users and it is possible for malicious parties to act as a CA and issue fake certificates. This can cause users to trust malicious websites and share their personal records and bank information with them. This problem can be mitigated with PDLs, by distributing trust between a group of users rather than a single entity and can be compromised only when more than 50% (or any higher threshold set by the governance) nodes are malicious. As soon as a user credential are allocated, the respective smart contract can be executed and all the relevant information for the certificate is recorded. These credentials may be used to access the controlled data or records (e.g. PDL data). Since, the credentials are issued by the group of users in a PDL and their integrity is backed by a transparent mechanism, so can be trusted. Also, it is difficult for malicious users to act as a CA because PDLs are managed by a group of nodes and all the records (such as public keys) are transparent, so the users can verify the integrity of a website with the PDL[add citation from PDL website].



Figure 7‑1 Example of PDL based Certificate Authority

Smart contracts can provide a mechanism for access data from a forgein ledger, by distributing authorised keys to the authenticated participants only, in this way, the participants will not need to ask for access keys repeatedly; the key distribution is recorded via smart contract to a PDL enabling the records to be updated automatically and transparent to all PDL members. This facilitates the future audit of the access records.

## Solutions

### Introduction

Smart contracts have some limitations such as scalability and immutability, which are already discussed in earlier sections. In this section, we provide the possible solutions to these inherent properties of smart contracts (e.g. Immtability and auto-execution). It is to be noted that in certain cases these properties cannot be eliminated but can be mitigated through design and planning.

### Scalablity

All the executions of a smart contract is recorded in a PDL and removing them from PDL is a very difficult and a costly process. Because smart contracts cannot be removed, unused and dormant contracts may live for eternity in the PDL costing PDL node resources. We present some potential solutions to manage the scalability problem of PDLs due to smart contracts specifically.

#### Check-point

The smart contracts can be installed on side-chains with a check-point to self-destruct after certain time. A side-chain can record the exsistance of the contracts in the master-chain before destruction. This can be achieved by introducing a *check-point (e.g. a specific date).* For example, a side-chain with with certain dealings between a telecom operator and a vendor, and once this contract is completed, the chain is destructed but final settlement transaction may be recorded in the master-chain.

#### Extensibility:

Smart contracts are immutable, however, they can be extended or revised by adopting the off-chain mechanism. That is to say that, the master contract is deployed in a master chain (or may be the side-chain acting as master chain) with the initalising clauses only, and include commands which call the logic contract. The logic contracts are separate contracts which may be installed on the same or different (e.g. master-chain or side-chain) PDL or may be installed off-chain (i.e. trusted datastructure). A sample architecuture for contract chaining is shown in Figure 6-3.

### Security of contracts:

Smart contracts are software and are not web-based, hence the traditional application layer security protocols (such as https) are not applicable to them. Incorrect information can activate smart contracts in a manner which may have negative impact on the ledger and its users. A possible solution is mandating that activation requests for smart contracts are always generated from a Trusted Execution Envoirnment (TEE) (Figure 6). In Figure 6, a smart contract based QoS monitoring system is explained where TEE is installed on both the user and the operator, the request to execute a smart contract is generated from the user, however the QoS parameters are reported to the PDL through a TEE which is submitted to the operator through customer’s TEE. The detailed procedure is explained in next section.

### Example: Smart contracts with QoS monitoring

The architecture explained here, provides the mechanism of resource allocation using smart contracts that can be adopted by industry for smart contract coupled resource allocation.



Figure 7‑2: Smart contracts with QoS monitoring

1. All the contracts from all available operators are advertised on a DApp where the customer can input their requirements and choose a suitable network service offer. The customer is required to forward the agreed upon payment to the chosen operator using traditional means to prevent DDoS attack on the PDL.
2. Once a suitable offer is chosen, the web app fills an activation request to the corresponding contract, transferring the payment due at the same time (which will be actually carried only if the request is successful). This request is encoded as a transaction to then need to be signed by the customer, to prove their approval, and then sent to the PDL. Do note how all service offers are backed by already deployed SCs on the PDL to be able to accept requests. As such the SC deployment fees are paid by their operatord (i.e. PDL members with operator role), while activation request fees are paid by the customers requesting them.
3. The new transaction containing the activation request is added to the pool of pending requests by the validators (i.e. operators and regulatory authorities), that will eventually accept it, through the distributed consensus algorithm, if well formed.
4. On successful execution, the respective operator gets notified and can start allocating the resources to provide the requested service.
5. The Service between the operator and the customer is being provided. At this stage, the actual QoS is managed by the Trusted enclave (i.e. TEE) and the Edge Monitor (i.e. TEE).
6. The Edge Monitor records the receipts for the user back to the PDL inside the corresponding smart contract (i.e. their agreement). This allows to verify, and prove, if the SLA has been fulfilled.

## Needs - Requirements to build a viable system with Smart contracts

### Regulatory Aspects

The PDLs' governance should manage the smart contracts, the group organizing a PDL can reach to a consensus on the regulation of the terms and penalties in case of violation. For example, roaming is currently a challenge for mobile network operators. In current systems, billing in roaming may be a long process and involves several steps such as sending the usage to the home operator to make claims. To resolve this, customers’ payment can be directed to the visiting operators through the PDL and invocation of a smart contract. This system is only viable when both the participants honor the smart contract and in the situations of dispute resolve them as per the governance of the PDL.

### Security of the Contracts

In Blockchains such as Ethereum, smart contracts are publically available; as per Ethereum consensus a copy of every contract is stored at every node; this may not be a scalable strategy for many real-world applications, where all the participants, even from the same PDL are not involved in every agreement or contract.

In such situation, a more exclusive mechanism should be adopted, where only the involved participants, may have access to smart contracts. To ensure privacy in a smart contracts, different access rights can be assigned to every participant of the contract. Here, the participants can be direct trading parties or the other stakeholders such as the mediators (in PDL access control mechanisms may prevent security breaches). Another advantage of this strategy is it enables scalability for the nodes.

#### Secure data feed(oracles):

Smart contracts usually get data from external sources such as oracle services; sometimes, this data-feed is used by them to start executing specific functions such as payments and penalties. For example, in the telco-sector, the QoS records are submitted to a contract to perform payment functions for the network services provided. It is likely that the participants, such as clients, can tamper with the actual data to benefit themselves. For example, they report wrong QoS metrics to blame the provider for not offering the contractual service. This problem should be tackled at the implementation stage, however, security mechanisms such as installation of trusted hardware at the customer end for example Trstued Code Base/Trusted Execution Envoirnments(TEE) can be adopted after checking implications.

### Enforceability

Smart contracts are self-executable, which means they can automatically execute with the fulfilment of a certain pre-coded condition; When two or more parties internally or externally agree on a contract, they are expected to honour the agreement without any disputes and if there is any, the stakeholders can come together to resolve the issue as per organisation policies.

Smart contracts are enforceable across the borders (i.e. internationally) and should follow the PDL governance policies and the participants laws.[read the UN regulations for international trade[park for future discussion]

### Availablity

As smart contracts are aiming to be adopted as a contract mechanism for industry, an important consideration for them is to be always available for execution which depends on the transaction speed of the native PDL. If a PDL supports higher transaction speed it also allows more connections to smart contracts; the number of requests at the PDL (i.e. transactions) impacts the availability of the PDL hence smart contracts.

Hence, to avoid unwanted traffic at the PDL, admission control mechanisms may be applied to ensure legitimate and necessary nodes access the PDL only. For example, a PDL governance may enforce a rule to allow a node to send certain number of transaction in a specified time only, after they are not allowed to send transactions for some specified time or they should go to hibernation state that is their “Idle-time”. The number of allowed transactions and idle time of nodes depends on the use-case and the governance of the PDL, for example, a organisation using a PDL-type which allows hundreds of transactions per second may allow more frequent transactions from their users compared to other PDL-type which support tens of transactions per second which can accommodate less number of participants.

### Attacks

#### Reentrancy

Reentrancy attack happens when the attacker takes hold of the contract and attempts to change the ledger through this contract, one consequence is that they possibly able to transfer funds to themselves. The most famous example of this attack the DAO attack in 2016, in which the attacker was able to steal 3.6 million Ethers through a reentrancy attack.

Reentrancy can be 1) Single function and 2) Cross-function. In Single Function reentrancy, the attacker can control only one function and recursively calls the same function to create damage; for example, drain all funds managed by the contract. In Cross-function reentrancy, the attacker can control functions which share states with other functions. For example, a payout contract shares its state with a vulnerable function.

#### Free Option Problem

This type of problem is well discussed in Plasma blockchain. When two parties, X and Y agree to do some purchase and decide to pay through a smart contract, X sends its signed transaction; in the mean-time Y changes its mind and backs-off. In this situation X has already sent Y the payment for the item but Y has refused to send the product; in this case Y has the Free-Option he can take the money without giving the product. In PDLs, this type of attack can be mitigated by the governance and the penalities enfored by them.

#### Denial of Capacity Attack:

Like other distributed systems, PDLs are vulnerable to attacks from malicious parties which can cause Denial of Service (DoS) to legitimate users. For example, since the PDLs allow a finite number of trascactions per second, the malicous users can send continuous and redundant service requests from malicious users to the PDL which can overwhelm the PDL. A global lock of certain time(possibly few seconds) should be applied to prevent such happenings. Also, penalities through governance may also prevent such wrongdoings.

#  Limitations of Smart Contracts

##  Inter and Intra system threats

### Introduction:

 ITU[[3]](#footnote-4) in its report on DLT and finanacial .. identified these potential risks to smart contract technology: *1) A reliance on a computer system itself that executes the contract, 2) flaws in the smart contract code( discussed in section 6) , 3) or the reliance on an external ‘off-chain’ event or person – to integrate with and execute – the embedded terms of the contract.*

We discuss these three main points and provide their possible mitigation techniques.

### Absence of Termination clause/Self-Destruction

In every smart contract, a termination function is a fragile entity. If it does not exist or not programmed with the utmost care, can be active for an indefinite period, which can prove very dangerous. For example, if a contract is meant to be writing vehicle service records to the ledger such as location etc., and this car is sold by the company to another compnay, the absence of or flaw in termination function can result in this vehicle to continue sending the critical data to the ledger. This is dangerous to the new owner of the car because his information, perhaps critical, is being seen by a third-party; also, for the old owner as this vehicle is still utilising the ledger and occupying the costly storage. For example, if a contract stipulates a payment for a certain period of time and the contract does not expire after that period, the amount will be paid indefinitely. Indeed, the payments can be cancelled by other means such as informing banks to stop the payment but that is alse dependent on the design of the contract. Moreover, if such errors go unnoticed, can potentially result in bigger losses such as execution of certain terms which may harm the company’s reputation.

### Admission Control

Smart contracts should be allowed by authorized participants only through stringent access control mechanisms; a strong governance can potentially handle this, and consensus agreed by the PDL members. If smart contracts' access is not carefully managed, they can become open to malicious users. However, this risk in a PDL is minimum since the participants are usually known and allowed with consensus, yet the risk of a replay attack exists. In such attacks, the malicious party intercepts the communication, and sends a modified data; if an attacker can alter the data such as payment amount or the payee, the payments will be issued by the contract. Admission control mechanisms should ensure that the transactions received by the legitimate client only.

### Off-chain and side-chain contracts handling

Mostly Smart contracts are deployed in a PDL and may be recorded off-chain or on side-chain; when they are recorded off-chain or on side-chain some details of the contracts are still recorded to the master ledger such as the completion date or status of the contract. In this way the external parties (I,e, ) have certain access to the master-ledger. This can be risky in case the external ledger is compromised because they have certain access to the master ledger. Errorounous contracts if allowed in a PDL can kickoff transactions, these contracts can be on-chain or off-chain. Strong accountability mechanism is required to pinpoint which entity is compromised.

### Poor Exception Handling

If syntax and logic errors in a smart contract are not thoroughly checked and handled, it can cause an infinite loop or hanged contract; this danger can be mitigated by careful design and testing of contracts, as discussed in Section 5.

### Transparency of a PDL

Though private, PDL is still shared among members means that transactions are visible to the members. This can be dangerous when competators are sharing a ledger, for example in the situation of bidding, the price of bid is recorded as transaction in the ledger, the competing members can see this value in the ledger and can exploit this vulnerability. This situation can be migitated by governance such as using hash instead of actual value or enter only encrypted values in the ledger.

### External Libraries

Computer software such as smart contracts rely on built-in programming language libraries; these third-party libraries are prone to error, and using them may be risky. Furthermore, the malicious party can develop such a library to penetrate in smart contracts. Developers should be careful while using third-party libraries to avoid any dangers to the smart contract.

##  Limitations

### Introduction

Smart contracts’ inherent properties also cause some limitations. This section outlines these limitations and considerations that need management before the deployment of a viable contract. These limitations are specific to the smart contract and dependent on the underlying PDL-type. For example, if some PDL-type with high transaction speed, more smart contracts will be executed per second than the PDL-type, allowing fewer transactions per second.

### Occupancy:

Smart contracts are software codes, and they are installed on a PDL, which by-definition is immutable. Hence if a smart contract is installed on a PDL, it cannot be deleted or amended. As we discussed in earlier sections, there exist mechanisms that allow the contracts to be updated. With such techniques, a new copy of a smart contract is installed, then the pointer to the old contract is updated. These techniques do not remove the old contract, and it lives in the ledger but dormant.

If dormant and inactive contracts populate a PDL, it can cause scalability problems over time.

### Latency

The key consideration for deploying a smart contract is the delay or latency. The latency of a smart contract is the time it takes for a contract to get deployed and executed and can be categorised in 1) deployment latency and 2) execution latency.

Smart contracts get compiled on the local machines which can potentially be personal computers; then the request to deploy them is issued by the deployment entity through a transaction. In this situation, the smart contract latency is dependent on the compilation of the code and the network delay for a contract request to reach the chain.

Mostly, the Smart contracts get executed more often then deployment. the pre-deployed smart contract can be executed by any entity with the right permissions. To execute or invoke a smart contract a transaction is issued by the invoking entity and this depends upon the factors such as network connection and the congestion at the chain. Moreover, the nodes of the ledger by-design are distributed across the World and computation and speed limitations of every node adds an overhead to the latency in verification of contract transaction.

The method of deployment and execution discussed here is a high-level picture of the smart contract system and is strongly dependent on the underlying chain.

### Underlying and Relying ledgers in permissioned context:

One of the most important considerations for the industry to adopt smart contract technology is that of the underlying ledger. Smart contracts are deployed on the ledger such as Corda, Ethereum or Hyperledger Fabric. Every ledger is unique in its properties and has different resource requirements. As of the time of writing this report, there is no system for ledgers to interact with different ledger exist, all the organizations or nodes must use the same underlying ledger technology in order to implement the smart contract as their contractual mechanism. This is not always possible for several reasons such as economically and feasibly to use same ledger technology; hence, be part of the consortium.

### Not every term can be translated to a Smart Contract

Smart contracts are nonetheless a computer program, and computer programs have very strict rules, such as if this then that or do this until this condition becomes true or false. But, in real-world contracts, the conditions are not always this rigid and there are leeways allowed intentionally by both parties, for example, if a business relationship between two organisations is old and they do want to give each other some discount but not to record in the contract, then it may be difficult to have a smart contract in this situation. For a smart contract, either it is or it is not, there is no opportunity for a middle ground. However, it is important for parties to be transparent in the contractual process and such biletral promoises which cannot be translated to the code, can be recorded in additional contract field in a plain text or in hash format, this will enable transparency between the participants. Adding this field in a hash form, can be verified later.

### Legal Uncretainity:

PDL are distributed nodes, which can potentially be spreaded across the globe. The enforceability of smart contracts in different countries can be a issue.

Legal aspects of contracts is beyond the scope of this report but geographic regulations and laws such as GDPR still applies and depends on the governance and consortia.

For example, if two parties are exists in same country, the country laws will apply but in multi-judridictional transaction it is recommended to follow the UNCITRAL arbitration rules and considerations.

### Intellectual Property Rights:

If a smart contract is deployed on a consortia ledger, the parties must be aware of the potential exposure of smart contract code to other parties, as depending on the ledger, either the source code of the contract, or the compiled version of the contract is shared across the whole distributed ledger. This requires the parties to manage IPR related to the smart contract, and potentially include licensing across consortia members or non-disclosure claims in the consortia agreement to meet the required IPR management standards across the different organizations.

### Accountability in Smart Contracts:

Following some pre-auditing mechanism to granted the completeness of availability of the Smart contracts, there would be two dimensions:

a)     Smart Contracts that are minimal functionalities or security functionality components with the building blocks consensus.

b)    Smart Contracts that are for business layers and for development and enhancement proposals.

In terms of functional components, the accountability has cleared by the governance model which should include a mechanism of testing, discoverability issues and mitigation of bugs before the genesis of the PDL which normally occur on testnet period before the mainnet, but it has to be audited before the genesis block of the network which it would be governed.

For the business layers there are a variety of approach which in permissioned environment, either public or private, have a modular ingredient whereby minimal terms of use are recommended and complete acquaintanceship with the the governing body of the PDL however in some cases the accountability could be a private permissioned environment whereby the responsibility and liabilities would be by the perfected interest in business although replicate the usage of the PDL in accordance with the consensus mechanism.

# Annex A:Title of annex

Annex B:
Title of annex

# B.1 First clause of the annex

## B.1.1 First subdivided clause of the annex

Annex C:
Bibliography

Annex :
Change History

| Date | Version | Information about changes |
| --- | --- | --- |
| 11-2019 | 0.0.1 | Initial draft – added table of contents |
| 12-2019 | 0.0.2 | Added SC introfuction, types and some text |
| 01-2019 | 0.0.3 | Cleaned draft  |
| 02 - 2020 | 0.0.4 | Cleaned up after F2F meeting – deleted section 6  |
| 03- 2020  | 0.0.5 | Added life cycle + section 9 |
| 06-2020 |  0.0.6 |  Limitations+cleared after F2F(online) meeting |
| 06-2020 | 0.0.6 | Added architecture |

# Document History

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

1. <https://dl.acm.org/doi/pdf/10.1145/3243734.3243780> [↑](#footnote-ref-2)
2. <https://dl.acm.org/doi/pdf/10.1145/3194113.3194115> [↑](#footnote-ref-3)
3. <https://www.itu.int/en/ITU-T/focusgroups/dfs/Documents/201703/ITU_FGDFS_Report-on-DLT-and-Financial-Inclusion.pdf> [↑](#footnote-ref-4)