



Baseline Core Version 1.0 - Part 1

Project Specification Draft 01

17 April 2024

This stage:

<https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/psd01/baseline-core-v1.0-psd01-part1.md> (Authoritative)

<https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/psd01/baseline-core-v1.0-psd01-part1.html>

<https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/psd01/baseline-core-v1.0-psd01-part1.pdf>

Previous stage:

N/A

Latest stage:

<https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/baseline-core-v1-part1.0.md> (Authoritative)

<https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/baseline-core-v1-part1.html>

<https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/baseline-core-v1-part1.pdf>

Open Project:

[Baseline](#), an initiative of [Ethereum Community Projects](#)

Project Chair:

Dr. Andreas Freund (a.freundhaskel@gmail.com)

Former Chair:

John Wolpert (john.wolpert@mesh.xyz), [ConsenSys Mesh](#)

Editors:

Dr. Andreas Freund (a.freundhaskel@gmail.com)

Kyle Thomas (kyle@provide.services), [Provide](#)

Yoav Bittan (yoav.bittan@mesh.xyz), [ConsenSys Mesh](#)

Keith Salzman (keith.salzman@mesh.xyz), [ConsenSys Mesh](#)

Chaaals Neville (chaals@entethalliance.org), [EEA](#)

Related work:

This specification is related to:

[baseline-api-v1.0] *Baseline API and Data Model Version 1.0 - Part 2* Edited by Dr. Andreas Freund, Yoav Bittan, Keith Salzman, Chaaals Neville, Anais Ofranc and Kyle Thomas. 17 April 2024. Project Specification Draft 01. <https://docs.oasis-open.org/ethereum/baseline/baseline-api/v1.0/psd01/baseline-api-v1.0-psd01-part2.md> . Latest stage: <https://docs.oasis-open.org/ethereum/baseline/baseline-api/v1.0/baseline-api-v1.0-part2.md>

[baseline-ccsm-v1.0] *Baseline CCSM Requirements Version 1.0 - Part 3* Edited by Dr. Andreas Freund, Yoav Bittan, Keith Salzman, Chaaals Neville, Anais Ofranc and Kyle Thomas. 17 April 2024. Project Specification Draft 01. <https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/psd01/baseline-core-v1.0-psd01-part3.md> . Latest stage: <https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/baseline-core-v1.0-part3.md>

Abstract:

This document describes the minimal set of business and technical prerequisites, functional and non-functional requirements, together with a reference architecture that when implemented ensures that two or more Systems of Record can synchronize their system state over a Consensus Controlled State Machine (CCSM) network with little or no trust assumptions.

Status:

This document is a Project Specification Draft and is no longer under active development.

This was last revised or approved by Baseline, part of the Ethereum OASIS Open Project, on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document.

Comments on this work can be provided by opening issues in the project repository or by sending an email to the project's public comment list baseline@lists.oasis-open-projects.org.

Keywords:

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119](#) when, and only when, they appear in all capitals, as shown here.

Citation format:

When referencing this specification the following citation format should be used:

[baseline-core-v1.0] *Baseline Core Specification Version 1.0 - Part 1* Edited by Dr. Andreas Freund, Anais Ofranc and Kyle Thomas. 17 April 2024. Project Specification Draft 01. <https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/psd01/baseline-core-v1.0-psd01-part1.md> Latest stage: <https://docs.oasis-open.org/ethereum/baseline/baseline-core/v1.0/baseline-core-v1-part1.0.md>

Non-Standards Track Work Product

Notices

Copyright © OASIS Open 2024. All Rights Reserved.

Distributed under the terms of the OASIS [IPR Policy](#).

For complete copyright information please see the Notices section in the Appendix.

Table of Contents

[1 Introduction](#)

[1.1 Overview](#)

[1.2 Glossary](#)

[1.3 Typographical Conventions](#)

[1.3.1 Requirement Ids](#)

[2 Design and Architecture](#)

[2.1 Agreement](#)

[2.2 State Object](#)

[2.3 Transacting Counterparties](#)

[2.4 Commercial Agreements](#)

[2.4.1 Contract](#)

[2.4.2 Commercial Documents](#)

[2.5 Consensus Controlled State Machine](#)

[2.6 Baseline Protocol Instance](#)

[2.7 High-Level Functional Requirements](#)

[2.7.1 Functional Requirements on commercial counterparties](#)

[2.7.2 CCSM Lifecycle Processes](#)

[2.8 Baseline Protocol Reference Architecture](#)

[2.8.1 State Synchronization](#)

[2.8.2 Considerations on BPI and CCSM Abstraction Layers and the CCSM Layer](#)

[2.8.3 External Applications](#)

[2.8.4 Baseline Protocol Stack Detailed Reference Architecture Layers and Components](#)

[3 Identifiers, Identity and Credential Management](#)

[3.1 Introduction and High-Level Requirements](#)

[3.2 BPI Identifiers, Identities and Credentials, and their Management](#)

[3.2.1 BPI Identifiers](#)

[3.2.2 BPI Identities and Credentials](#)

[4 BPI Abstraction Layers](#)

[4.1 BPI Abstraction Scope and Components](#)

[4.2 BPI Abstraction Layer Security and Integration](#)

[5 Middleware, Communication and Interoperability](#)

[5.1 BPI Subject Capabilities](#)

[5.2 BPI Subject Account](#)

[5.3 BPI Service Orchestration](#)

[5.4 BPI Communication](#)

[5.5 BPI Integration](#)

[5.5.1 Resolvable Identifiers for BPI Interoperability Services](#)

[5.5.2 Discoverable Authentication and Authorization Capabilities for BPI Interoperability Services](#)

[5.5.3 Discoverable and Negotiable BPI Interoperability Services](#)

[5.5.4 Bi- and Multi-directional and Mono-directional BPI Interoperability Services](#)

[5.5.4.1 Mono-directional BPI services](#)

Non-Standards Track Work Product

[5.5.4.2 Bi- or Multi-directional BPI services](#)

[5.6 Standardized Set of BPI Interoperability APIs](#)

[5.7 BPI Interoperability: Discoverable Standard Transport Security](#)

[6 Agreement Execution](#)

[6.1 BPI Workstep](#)

[6.2 BPI Workflow](#)

[6.3 BPI Workgroup](#)

[6.4 BPI Account](#)

[6.5 BPI Transactions](#)

[6.6 BPI Transaction Lifecycle](#)

[7 General BPI Storage Capabilities](#)

[7.1 BPI Storage Security](#)

[7.2 BPI Storage Privacy](#)

[7.3 BPI Data Orchestration](#)

[7.4 BPI-External Storage: Edge Storage](#)

[7.5 BPI-Internal Storage](#)

[7.5.1 BPI Storage: Centralized Deployment](#)

[7.5.2 BPI Storage: Decentralized Deployment](#)

[8 BPI External Data Inputs](#)

[8.1 Internal Authoritative Data for BPIs](#)

[8.2 External Authoritative Data for BPIs](#)

[8.3 External Non-authoritative, Non-deterministic Data for BPIs](#)

[8.3.1 Data Trustworthiness](#)

[8.3.2 External Non-authoritative, Non-deterministic BPI Input Data Variance](#)

[9 Conformance](#)

[9.1 Conformance Targets](#)

[9.2 Conformance Levels](#)

[Appendix A - References](#)

[A.1 Normative References](#)

[A.2 Non-Normative References](#)

[Appendix B - Security Considerations](#)

[B.1 Data Privacy](#)

[B.2 Production Readiness](#)

[Appendix C - Acknowledgments](#)

[Appendix D - Revision History](#)

[Appendix E - Notices](#)

1 Introduction

The Baseline Protocol is an open-source initiative that combines advances in cryptography, messaging, and Consensus Controlled State Machines (CCSMs) often referred to as blockchains or distributed ledger technology (DLT) to deliver secure and private business processes at low cost – event ordering, data consistency, and workflow integrity. The Baseline Protocol provides a framework that allows Baseline Protocol Implementations (BPIs) to establish a common (business) frame of reference enabling confidential and complex (business) collaborations between enterprises without moving any sensitive data between traditional Systems of Record. The work is an [Ethereum Community Project](#), which is managed by [OASIS](#).

1.1 Overview

An illustrative example of the use of a BPI is a Buyer placing an order to a Seller. Normally a Buyer system creates an Order and transmits it to the Seller system through some preestablished messaging system without providing any proof that the Order created is correct, forcing the Seller's systems to validate the order, and more often than not, finding data inconsistencies between the Seller system and the Order. This then leads to a time-consuming, and often expensive, back and forth between Seller and Buyer to rectify the issue.

In the case that a BPI is used, the Buyer action of creating an order and submitting it to the BPI creates a cryptographic proof on the BPI that the order conforms (or not) to the agreed-upon commercial contract terms and current contract state between Buyer and Seller stored on the BPI, whereupon verification, the commercial contract state on the BPI is updated based on the order details. Subsequently, the cryptographic proof of order correctness is attached to the order and sent to the Seller using either established integrations or the BPI. The Seller can then directly validate the proof without having to check the correctness of the Order against its System of Record anymore. A valid cryptographic proof ensures that the order will be correctly formulated the first time avoiding errors, and thus saving time and money – a more detailed example is provided in section [2.8.1 State Synchronization](#). A BPI, therefore, enforces the synchronization of Systems of Record between Buyer and Seller.

Non-Standards Track Work Product

At a high level a BPI's benefits and characteristics can be summarized as follows:

- BPI usage avoids rework between contract counterparties due to improperly applied business logic because cryptographic proofs of correctness ensure that Systems of Record remain synchronized, and that rework is minimized or even completely avoided.
- Existing integrations can continue to be used with only minimal augmentation of cryptographic proofs that contractual business logic and data such as discounts are properly applied.
- Service request validation such as that of an order against the service request receiver's System of Record is no longer necessary.

In the following, this document lays out the requirements on a BPI to achieve above described benefits and characteristics in several sections:

- **Section 2: Design and Architecture** with definitions, key concepts, and overviews of the components of a compliant Baseline Protocol Implementation as delineated in the following subsections:
 - State Object
 - Transacting Counterparties
 - Commercial Agreements
 - Consensus Controlled State Machine (CCSM)
 - Baseline Protocol Instance
 - High-Level Functional Requirements
 - Baseline Protocol Reference Architecture
- **Section 3: Identifiers, Identity and Credential Management** with definitions, key concepts, and overviews of identifiers, identities, and credentials necessary in a compliant Baseline Protocol Implementation as delineated in the following subsections:
 - Introduction and high-level Requirements
 - BPI Identifiers, Identities and Credentials, and their Management
- **Section 4: BPI Abstraction Layers** with definitions, concepts, scope, and security considerations for BPI Abstraction Layers:
 - BPI Abstraction Scope and Components
 - BPI Abstraction Layer Security and Integration
- **Section 5: Middleware, Communication, and Interoperability** with definitions, key concepts, and overviews of BPI Subject Capabilities and Accounts, service orchestration, communication, BPI Interoperability with its APIs, and Transport Security:
 - BPI Subject Capabilities
 - BPI Subject Account
 - BPI Service Orchestration
 - BPI Communication
 - BPI Integration
 - Standardized Set of BPI Interoperability APIs
 - BPI Interoperability: Discoverable Standard Transport Security
- **Section 6: Agreement Execution** with definitions, key concepts, and overviews of the BPI Processing Layer components necessary in a compliant Baseline Protocol Implementation as delineated in the following subsections:
 - BPI Workstep
 - BPI Workflow
 - BPI Workgroup
 - BPI Account
 - BPI Transactions
 - BPI Transaction Lifecycle
- **Section 7: General BPI Storage Capabilities** with definitions, key concepts, and overviews of BPI Storage components applicable to all BPI layers and necessary in a compliant Baseline Protocol Implementation as delineated in the following subsections:
 - BPI Storage Security
 - BPI Storage Privacy
 - BPI Data Orchestration
 - BPI-External Storage: Edge Storage
 - BPI-Internal Storage
- **Section 8: BPI External Data Inputs** with definitions, key concepts, and overviews of BPI External Data Input components necessary in a compliant Baseline Protocol Implementation as delineated in the following subsections:
 - Internal Authoritative Data for BPIs
 - External Authoritative Data for BPIs
 - External Non-authoritative, Non-deterministic Data for BPIs
- **Section 9: Conformance** with a specification of conformance tests for each requirement and definitions of the different levels of conformance of a BPI to this standard.
 - _ Conformance Targets
 - _ Conformance Levels

1.2 Glossary

Anti-Money Laundering:

Anti-money laundering (AML) refers to a set of laws, regulations, and procedures intended to prevent criminals from disguising illegally obtained funds as legitimate income.

International Monetary Fund, Reference Guide to Anti-Money Laundering and Combating the Financing of Terrorism Second Edition and Supplement on Special Recommendation IX, 2006.

Baseline Protocol:

The Baseline Protocol is a set of methods that enable two or more state machines to achieve and maintain data consistency, and workflow continuity by using a network as a common frame of reference.

Baseline-Bridge:

A mechanism for one Workflow to use the proof generated by a different Workflow.

For instance a proof generated in a Workflow executed by Workgroup A to be used as input to a Workflow executed by Workgroup B.

Baseline-Connector:

An interface connecting and synchronizing a baseline stack and system of record.

Byzantine Fault Tolerant (BFT):

Given a network or system of n components, t of which are dishonest, and assuming only point-to-point channels between all the components, then whenever a component A tries to broadcast a value x such as a block of transactions, the other components are permitted to discuss with each other and verify the consistency of A 's broadcast, and eventually settle on a common value y . The system is then considered to resist Byzantine faults if a component A can broadcast a value x , and then:

- If A is honest, then all honest components agree on the value x .
- If A is dishonest, all honest components agree on the common value y .

"The Byzantine Generals Problem", Leslie Lamport, Robert E. Shostak, Marshall Pease, ACM Transactions on Programming Languages and Systems, 1982.

Circuit Breaker:

The ability of a Party to immediately cease all their active Workflows across all of their Workgroups within a Baseline-compliant implementation, and, if required, exit a Baseline-compliant implementation with all their data without any 3rd party being able to prevent the exit.

Non-Standards Track Work Product

Common Frame of Reference:

A Common Frame of Reference as used in this document refers to achieving and maintaining data consistency between two or more Systems of Record using a consensus-controlled state machine. This enables workflow and data continuity and integrity between two or more counterparties.

Consensus Controlled State Machine:

A Consensus Controlled State Machine (CCSM) is a network of replicated, shared, and synchronized digital data spread across multiple sites connected by a peer-to-peer and utilizing a consensus algorithm. There is no central administrator or centralized data storage.

Data Orchestration:

An automated process for taking siloed data from multiple storage locations, combining and organizing it, and making it available for analysis.

Electronic Record:

Information captured through electronic means, and which may or may not have a paper record to back it up.

Bulletin of the American Society for Information Science and Technology, Electronic Records Research Working Meeting: A Report from the Archives Community, May 28-30, 1997.

Identity:

The condition of being the same with something described or asserted, per Merriam-Webster Dictionary.

A concretization of the above used in this document: Identity is the combination of one or more unique identifiers with data associated with this/these identifier(s). Identity-associated data consists of signed certificates or credentials such as verifiable credentials and other unsigned, non-verifiable data objects generated by or on behalf of the unique identifier(s).

[Merriam-Webster Dictionary](#)

Interoperability:

The ability of a Party operating Workflows on a baseline-compliant implementation A to instantiate and operate one or more Workflows with one or more Parties on a baseline-compliant implementation B without the Parties on either implementation A or B having to know anything of the other Parties' implementation.

Legal Entity:

An individual, organization, or company that has legal rights and obligations.

Liveness:

In concurrent computing, liveness refers to a set of properties of concurrent systems that require a system to make progress, despite its concurrently executing components ("processes") may have to "take turns" in critical sections, parts of the program that cannot be simultaneously run by multiple processes. Liveness guarantees are important properties in operating systems and distributed systems.

Alpern B, Schneider FB (1985) Defining liveness. Inf Proc Lett 21:181-185

Low latency:

A latency that does not Materially Impact the overall system latency of the BPI.

Master Services Agreement (MSA):

A legal contract that defines the general terms and conditions governing the entire scope of products commercially exchanged between the parties to the agreement.

Material Impact:

In the context of certain requirements, such as [\[R96\]](#), something that causes the underlying business requirements of the BPI not to be met. For example in some deployment situations, a 5-second delay can cause transactions to fail or introduce instability to the system, while in other circumstances a 5-minute delay in processing makes no difference to the system as a whole.

Non-Repudiable:

Refers to a situation where a statement's author cannot successfully dispute its authorship or the validity of an associated contract. The term is often seen in a legal setting when the authenticity of a signature is being challenged. In such an instance, the authenticity is being "repudiated".

Party:

An entity participating in the execution of one or more given Workflows within a Workgroup. A Workgroup is set up and managed by one Party that invites other entities as Parties to join as workgroup participants.

Portability:

The ability of a Party to migrate and re-baseline its existing Workflows and data from one baseline-compliant implementation to another baseline-compliant implementation without any 3rd party being able to prevent the migration.

Principal Owner

An entity controlling the public key(s) which control the identity and its identifiers.

Privacy Assurance Mechanism:

A way of ensuring the privacy of Workflow data represented on a consensus controlled state machine (CCSM) network.

Proof of Correctness:

A Proof of Correctness is a mathematical proof that a computer program or a part thereof will, when executed, yield correct results, i.e. results fulfilling specific requirements. Before proving a program correct, the theorem to be proved must, of course, be formulated. The hypothesis of such a correctness theorem is typically a condition that the relevant program variables must satisfy immediately before the program is executed. This condition is called the precondition. The thesis of the correctness theorem is typically a condition that the relevant program variables must satisfy immediately after the execution of the program. This latter condition is called the post-condition. The thesis of a correctness theorem may be a statement that the final values of the program variables are a particular function of their initial values.

"Encyclopedia of Software Engineering",
Print ISBN: 9780471377375| Online ISBN: 9780471028956| DOI: 10.1002/0471028959,
(2002), John Wiley & Sons, Inc.

Succinct:

Verification of a zero-knowledge proof by any 3rd party in a time that is sublinear to the size of the prover system that generated the proof.

System of Record:

Non-Standards Track Work Product

The integrity of the data in data architecture is established by what can be called the "system of record." The system of record is the one place where the value of data is definitively established. Note that the system of record applies only to detailed granular data. The system of record does not apply to summarized or derived data.

W.H. Inmon, Daniel Linstedt and Mary Levins, "Data Architecture", 2019, Academic Press, ISBN: 978-0-12-816916-2.

Trust Model:

Collection of entities and processes that entities rely on to help preserve security, safety, and privacy of data and which is predicated on the use of a CCSM implementation.

Marsh S. (1994). "Formalizing Trust as a Computational Concept". Ph.D. thesis, University of Stirling, Department of Computer Science and Mathematics.

Verifiably Secure:

A Process that enables a computer to offload the computation of some function to other perhaps untrusted clients, while maintaining verifiable, and, thus, secure results. The other clients evaluate the function and return the result with proof that the computation of the function was carried out correctly. The proof is not absolute but is dependent on the validity of the security assumptions used in the proof. For example, a blockchain consensus algorithm where the proof of computation is the nonce of a block. Someone inspecting the block can assume with virtual certainty that the results are correct because the number of computational nodes that agreed on the outcome of the same computation is defined as sufficient for the consensus outcome to be secure in the consensus algorithm's mathematical proof of security.

Gennaro, Rosario; Gentry, Craig; Parno, Bryan (31 August 2010). Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. CRYPTO 2010 [doi:10.1007/978-3-642-14623-7_25](https://doi.org/10.1007/978-3-642-14623-7_25).

Workflow:

A process made up of a series of Worksteps between all or a subset of Parties in a given Workgroup.

Workgroup

A workgroup is a set of Parties, also referred to as BPI Subjects, who are the authorized users of a BPI. The Parties use Workflows to synchronize their Systems of Record through one or more worksteps in the workflow.

Workstep:

A workstep is characterized by input, the deterministic application of a set of logic rules and data to that input, and the generation of a verifiably deterministic and verifiably correct output. A set of Worksteps used to synchronize the outcomes in two or more Systems of Record within a Workgroup is called a Workflow.

1.3 Typographical Conventions

1.3.1 Requirement Ids

A requirement is uniquely identified by a unique ID composed of its requirement level followed by a requirement number, as per convention **[RequirementLevelRequirementNumber]**. There are four requirement levels that are coded in requirement ids as per below convention:

[R] - The requirement level for requirements which IDs start with the letter **R** is to be interpreted as **MUST** as described in [RFC2119](#).

[D] - The requirement level for requirements which IDs start with the letter **D** is to be interpreted as **SHOULD** as described in RFC2119.

[O] - The requirement level for requirements which IDs start with the letter **O** is to be interpreted as **MAY** as described in RFC2119.

Note that requirements are uniquely numbered in ascending order within each requirement level.

Example: It should be read that [R1] is an absolute requirement of the specification whereas [D1] is a recommendation and [O1] is truly optional.

2 Design and Architecture

This section provides definitions, key concepts, and overviews of the components of a Baseline Protocol Implementation compliant with the requirements of this document. This section intends to provide implementers with guidance to be able to build and operate implementations of the Baseline Protocol not only in an informal but also in a very formal, highly regulated context. The latter context in particular requires implementers to prove that compliance with this standard not only does not violate regulatory frameworks but rather enables meeting regulatory requirements throughout the entire stack.

The key concepts, definitions, and associated requirements underpinning a Baseline Protocol Implementation (BPI) which will be discussed in this section are:

- State Object
- Transacting Counterparties
- Commercially and Legally Binding Documents
- Consensus Controlled State Machine
- Baseline Protocol Instance
- High-Level Functional Requirements
- Baseline Protocol Reference Architecture

2.1 Agreement

An agreement is a manifestation of mutual assent by two or more parties to one another see [Cornell Law School](#). An Agreement between two or more Counterparties allows for transactions between them dealing with, for example, commercial items such as products. An Agreement, therefore, governs and defines all transactions between counterparties.

2.2 State Object

A State Object is an item that can be exchanged between counterparties to an agreement, the state of which the counterparties have agreed on, and which is defined in an agreement. In the context of this standard, a State Object is assumed to be a document derived from an agreement and representing a specific state of either an asset, a product offering or a service offering transacted between counterparties.

Examples include, but are not limited to, a:

- Quote
- Order
- Invoice referencing for example:
 - Digital Services
 - Physical Products
 - Financial Assets

2.3 Transacting Counterparties

A transacting counterparty, or simply counterparty, that requests one or more State Objects from another counterparty is the Requester concerning the State Object. The Requester can also be the final recipient of one or more State Objects.

A counterparty that provides one or more State Objects to another counterparty is the Provider concerning the provided State Objects. The Provider is accountable to the Requester for all the

Non-Standards Track Work Product

State Objects it provides to the Requester.

A Requester may request State Objects from multiple Providers, and in the context of a supply chain of State Objects, a Provider of one State Object may also play the role of a Requester of other State Objects.

[R1]

Transacting counterparties MUST have an agreement specifying minimally the transactions to be performed between the transacting counterparties before a transactable state of an agreement can be instantiated within a BPI.

Note, that an agreement in the context of this document may or may not be a commercial agreement as defined in [4-Commercial-Agreements](#) below.

[R1] Testability: Functional terms of an agreement between counterparties are always implementable as transactions within a Baseline Protocol system because they are logical constraints which can always be represented in code. For example, payment term of 15 days can be checked whether the payment invoice is larger or smaller than the payment term.

2.4 Commercial Agreements

If transacting counterparties, also being referred to as Requester(s) and Provider(s) ("the parties") in this document, intend to establish a business and operational environment for commercial transactions, they will sign one or more agreements that commercially bind the parties, also known as Commercial Agreements. The aforementioned is only to be understood within the context of this document.

Such commercial, and thus presumably legally enforceable, documents may be presented as a combination of one or more of the following:

- Master Services Agreement
- Specific Terms and Conditions
- Order

[R2]

The parties to a Commercial Agreement MUST have signed commercial documents with each other before a transactable state of a commercial agreement can be instantiated within a BPI.

[R2] Testability: A commercial agreement represents a state that has been agreed on in specific terms between two or more counterparties. Such an agreement with all its elements including its counterparty signatures can be implemented into a baseline protocol implementation (BPI) through a Merkle Tree structure stored in database where each agreement element represented a Merkle Tree leaf. Merkle Trees can be readily implemented. A secure, offchain Merkle Tree library with tests can be found [here](#).

2.4.1 Contract

This section details the prerequisites required to be fulfilled by a legal contract, also referred to as a Commercial Agreement in this document, between the parties, and defines the general terms and conditions in the legal contract governing commercial transactions between these parties. These prerequisites are to be understood only within the context of this document and are meant to be general and not specific to particular legal and regulatory frameworks.

A contract will typically govern all commercial transactions between its parties and includes, but is not limited to, sections defining the Governing Law, the Legal Jurisdiction, Indemnity, Liability, Force Majeure, Charges and Taxes, Term, Obligations, definitions of commercially relevant elements such as locations, equipment, and products, as well as any other terms and conditions that apply to the entire scope of commercial and legal relations between the parties. Other legal documents, such as an order, typically reference the contract for its general terms and conditions and might contain more specific terms and conditions, such as rates and discounts and other commercial information, relevant for the specific context of the legal document. These specific terms and conditions can expand or override the original contract and are intentionally not specified in the original contract. The contract is the legal document from which specific commercial documents, such as a Quote or an Order, are derived.

[R3]

There MUST be a legally binding contract, however simple and temporary, before a commercial transaction — such as an order — between parties takes place.

For example, the contract and the order can be combined into a single document for a single transaction. However, there must be a legal framework in place to provide context for monies that are exchanged and settled. The functional part of the contract forms the basis of a Baseline Protocol Implementation (BPI) defined in section [2.6 Baseline Protocol Instance](#). The requirements below are to be understood solely within the context of this document. They are not meant to be generalized beyond this context.

[R3] Testability: A legally binding contract between parties can be implemented in a baseline system referencing the suggested [Order example](#) as a W3C Verifiable Credential with its [test suite](#).

[D1]

The contract SHOULD be in an electronic form.

[D1] Testability: A legally binding contract between parties can be implemented in a baseline system referencing the suggested [Order example](#) as a W3C Verifiable Credential with its [test suite](#).

[D2]

The functional terms of the contract SHOULD be represented on a BPI between the counterparties.

For example, a Payment Term such as N30, a discount value, agreed upon product numbers, Service Level Agreements (SLAs) etc.

[D2] Testability: Legally binding functional terms of a contract between parties can be implemented in a Baseline Protocol Implementation using for example [a Merkle Tree](#), as well as the functional terms represented as a Zero Knowledge Circuit [Example](#).

[D3]

The contract SHOULD be an MSA between the contract parties.

An MSA is preferable since it allows a proliferation of contract-based BPI workflows and worksteps between the parties reducing complexity and potential errors.

[D3] Testability: MSAs between counterparties can be implemented in the same way as a legally binding contract between parties, referencing the suggested Merkle Tree [Example](#), as well as the functional terms represented as a Zero Knowledge Circuit [Example](#).

[CR1]<[D3]

There MUST be only one MSA between contract parties covering commercial transactions for a given set of products, services, or assets.

This requirement is intended to disambiguate which terms cover which part of a commercial relationship between parties.

Specific Terms and Conditions ("Specific T&Cs") defines the terms and conditions governing a specific product, service, or asset or set thereof offered and delivered by Provider(s) to Requester(s).

[CR1]<[D3] Testability: Business logic can be expressed in software code, and software code can be tested, and since BPI logic can ensure only unique combinations of an MSA, unique identifier, and an agreement type exists between two or more counterparties, the requirement is testable.

[D4]

Each specific product, service, asset, or set thereof offered and delivered by Provider(s) to Requester(s) SHOULD have its specific T&C document.

This would allow the fine-graining and consistent application of commercial State-Object-specific business rules and data.

Non-Standards Track Work Product

[D4] Testability: Since a T&C document is part of a contract and a contract can be represented in a testable manner per the testability statement of [R3], requirement [D4] is testable.

2.4.2 Commercial Documents

Commercial documents, a category of commercial State Objects, refer to the state of a specific product/service/asset or set thereof, which may or may not be modified from an original offering to meet the Requester requirements and includes operational and commercial details. A commercial document is an abstract construct representing mutual commitments based on a legally binding contract.

[R4]

A commercial State Object to be transacted on MUST be based on a specific commercial document.

[R4] Testability: Since the testability statement in [R2] represents a commercial agreement with commercial documents and is testable as stated, this requirement is testable as well.

[R5]

A commercial document MUST be derived from a legally binding contract.

[R5] Testability: A legally binding contract outside of a BPI can be translated into a commercial document on the BPI as given in this [example](#) of a legally binding contract expressed as a W3C Verifiable Credential which can be tested following the W3C Verifiable Credential test suite approach.

[R6]

A commercial document MUST be represented as an electronic record on a BPI between the counterparties.

[R6] Testability: Since the testability statement in [R5] represents an electronic version of a commercial document and is testable as stated, this requirement is testable as well.

[R8]

A commercial document MUST be authorized by legal representatives of the parties or their legal delegates.

[R8] Testability: A digital signature of an authorized user of the BPI state object associated with the commercial document can be extracted from the proof section of this [example](#), validated and compared to the key identified in the verification method.

[D5]

The definition of a commercial document authorization SHOULD be stated in the legal contract underlying the commercial document.

Authorizations for commercial transactions are a foundational element in the context of this document, as they are in paper based agreements. Therefore, any legal authorization agreements relevant to the commercial agreement between commercial counterparties, and thus to commercial transactions between them, are important to be represented in a BPI to ensure mitigating the risk of unauthorized signatures.

[D5] Testability: Given is an [a W3C Verifiable Credential Example](#) showing the definition of a commercial document authorization by identifying "the buyer" and "the issuer" of the contract. A test to validate the "buyer" and "issuer" requirements can be written following the W3C Verifiable Credential test suite approach.

[D6]

The representatives and their authorized delegates who can perform commercial document authorizations SHOULD be explicitly listed or inferred from the stated legal delegation rules of the counterparties in the contract underlying any commercial document.

[D6] Testability: Given is an [a W3C Verifiable Credential Example](#) showing the definition of a commercial document authorization by identifying "the buyer" and "the issuer" of the contract. A test to validate the "buyer" and "issuer" requirements can be written following the W3C Verifiable Credential test suite approach.

[R9]

A commercial document MUST be non-repudiable.

Note that while non-repudiation in the physical world is most often tied to a physical signature of a Legal Entity on a legal document, in the digital world a digital signature over a digital legal document such as an Order or an Invoice belonging to a known and verifiable digital identity of a counterparty serves the same purpose.

Example: A Buyer ("Requester") and Seller ("Provider") may agree that a signed Order requires a signed original paper copy, or a digitally signed electronic Order Form, in addition to an Order being digitally signed and recorded within a BPI.

[R9] Testability: The non-repudiation of a commercial document can be achieved through the verification of the digital signatures from both parties over the contract. An [example](#) of this is given, where in the "proof" section two digital signatures can be found.

2.5 Consensus Controlled State Machine

A Consensus Controlled State Machine (CCSM) is a network of replicated, shared, and synchronized digital data spread across multiple sites connected by a peer-to-peer and utilizing a consensus algorithm. There is no central administrator or centralized data storage.

A CCSM with no or limited trust assumptions is the foundational enabler of a BPI.

For specificity, the popular words "Blockchain" or "DLT" are a particular form of CCSM design.

2.6 Baseline Protocol Instance

Baseline Protocol Instances or Implementations (BPIs) are logical constructs shared between transacting counterparties of Requesters and Providers and implemented on a CCSM. They are used to either validate or reconcile transactions between Requesters and Providers related to all State Objects transacted between them. The nature of bi- or multi-lateral transactions is such that two or more parties may transact to/from each other interchangeably.

Abstractly, a BPI consists of:

- The private messaging between Agreement Counterparties about the state, or the requested or finalized state changes, of the State Objects between them
- The representation of an agreement and documents and their business rules and data as distinct workflows and worksteps between transacting counterparties organized into workgroups based on the stipulations of the agreement
- The deterministic processing and finalization of state change requests based on documents between the transacting counterparties as stipulated by the agreement
- The preservation of the privacy of all transacting counterparties and their data from other 3rd parties

BPIs are strongly dependent on the security and privacy capabilities of the CCSM used to implement a BPI because BPIs without a CCSM are a single point of failure, whereas with a CCSM, there is no longer a single point of failure for the state of a BPI because a CCSM has no single-point-of-failure by its very definition.

[R10]

A BPI MUST utilize a CCSM.

[R10] Testability: The implementation of a BPI using, for example, an [Ethereum Client Transaction Crafting Function](#) demonstrates how a CCSM transaction is created, signed and sent to a CCSM client, and is therefore utilizing a CCSM.

Non-Standards Track Work Product

Since security and privacy requirements of a BPI are key, and are strongly dependent on the security and privacy assurances the CCSM on which the BPI is implemented can provide, BPIs need to take great care to avoid the following two situations:

1. Weaken the security assurances of the underlying CCSM by increasing the CCSM attack surface. Such an expansion of the attack surface can occur through, for example, the concentration of value-at-risk in one or more BPIs above the value used to economically secure the underlying CCSM. This situation would provide an economic incentive to attack, and subvert, the underlying CCSM to extract the value in one or more BPIs. 2. Increase the existing attack surface of a CCSM such that the security assurances of the BPI become significantly weaker than the underlying CCSM. An example of such a situation can occur when a commercial State Object such as a Financing contract or an Order in BPI A is dependent on a commercial State Object such as an invoice as collateral in BPI B, and when BPI B has weaker transaction finality assurances than either BPI A or the underlying CCSM. In that scenario, the commercial State Object in BPI A cannot provably rely on the invoice as collateral in BPI B since the invoice might be reverted, and it would then no longer be suitable collateral.

Hence, this document enumerates the following requirements below:

[R11]

A BPI MUST have the same security assurances as to the CCSM it utilizes.

[R11] Testability: The three security assurances given by the CCSM (Data Immutability, Provable Time Linearization, and Double Spend Protection) are automatically extended to the BPI data and, therefore, the BPI itself, when a BPI commits the Zero Knowledge Proof, the Public Input of the Proof, and the New State Commitment to, as an example, the [Shield Smart Contract](#) on the CCSM because all the data to verify the BPI state are on the CCSM.

[R12]

A BPI MUST support cryptographic algorithms that have public libraries with verifiable security audits and are recommended by public security authorities such as the US National Institute of Standards and Technology (NIST).

For information, please refer to appendix [A.2 Non-Normative References](#) for the cryptographic libraries that successfully passed the NIST Cryptographic Module Verification Program [\[CVMP\]](#).

[R12] Testability: The [A.2 Non-Normative References](#) for the cryptographic libraries that successfully passed the NIST Cryptographic Module Verification Program [\[CVMP\]](#) are testable.

[R13]

If a BPI utilizes a Peer-to-Peer (P2P) message protocol, the protocol MUST support end-to-end encryption.

[R13] Testability: This can be accomplished through a number of protocols, examples of P2P suitable messaging protocols include libp2p, NATS, or DIDComm that offers the ability to encrypt messages, for example through onion encryption and routing where the relays use layers of encryption around the original encrypted payload. Using an encrypted JSON web token (JWE) for the original message payload is recommended.

[R14]

A BPI MUST support cryptographic key management incl. backup and recovery that adheres to established industry security standards such as the US Federal Information Processing Standard [\(FIPS\)](#) or [ISO 27001](#).

[R14] Testability: US Federal Information Processing Standard [\(FIPS\)](#) or [ISO 27001](#) are testable.

[R15]

State changes of a BPI MUST be verifiable on the CCSM it utilizes.

Verifiable in this context means that a 3rd party can verify, via a cryptographic proof on the CCSM, that a transaction changed the state of a State Object in the BPI correctly, based on agreed-upon business rules - for example changing the Order status from open to completed.

[R15] Testability: This can be achieved through the use of Zero-Knowledge Proof verification in a smart contract, such as this [example](#)

[D7]

A BPI SHOULD have at least the same Liveness properties as the CCSM it utilizes.

Liveness means that if a CCSM does not require counterparties to constantly monitor its state to ensure that the state of the CCSM is correct, then the BPI should not require constant observation of its state either.

[D7] Testability: For a BPI to have the same Liveness properties as the CCSM it utilizes, you could, for example, store the relevant business information such as the Zero Knowledge Proof, the Public Input, and State commitment in a smart contract on the CCSM, giving the stored data the CCSM Liveness property.

[R16]

A BPI MUST be censorship-resistant.

Censorship-resistant means that a transacting counterparty can terminate a transaction at any time without another transacting counterparty, or any Node of the CCSM used to implement the BPI, being able to stop the termination of the transaction.

[R16] Testability: A simple way to implement the described censorship resistance in a testable manner is to define a revocation commitment that allows undoing an in-process transaction as is for example specified in the [DID Sidetree specification](#) with its [test vectors](#).

[R17]

A BPI MUST be able to provide privacy of the transacting counterparties' data.

[R17] Testability: Asymmetric encryption of the data by encrypting to a shared key between BPI users and BPI requires both parties to decrypt the data, one party is not enough.

[R18]

A BPI MUST implement date, time and timestamps according to [IETF RFC 3339](#).

[R18] Testability: All requirements for [IETF RFC 3339](#) are testable.

2.7 High-Level Functional Requirements

This section describes the prerequisites and high-level general operational framework requirements:

- Functional Requirements on commercial counterparties
- CCSM-based Lifecycle Processes

2.7.1 Functional Requirements on commercial counterparties

This section states the commercial and operational functionalities required from commercial counterparties.

[R19]

Commercial Counterparties MUST ensure that utilized BPIs allow them to meet all required legal, compliance, and business reporting requirements as it relates to their BPI activities.

Non-Standards Track Work Product

This comprises, e.g., fraud or tax audit requirements based on commercial transactions on a BPI.

[R19] Testability: Legal, compliance, and business reporting requirements are always implementable based on commercial transactions within a Baseline Protocol Instance. Adherence to these requirements can be verified by third parties utilizing Zero Knowledge Proofs.

[R20]

Commercial Counterparties MUST support the Reference Architecture defined in section [2.8 Baseline Protocol Reference Architecture](#).

[R20] Testability: All requirements for [2.8 Baseline Protocol Reference Architecture](#) are testable.

[R21]

Commercial Counterparties MUST use the BPI APIs to transact on a commercial State Object – see the [specification of the BPI APIs](#)

An ability of a Requester to request products, services, or assets, in other words, commercial State Objects, through an instance of the Baseline Protocol's APIs do not necessarily imply the ability to provide products, services, and assets through an instance of the Baseline Protocol APIs and vice versa.

Commercial counterparties need to know the level of conformity other commercial counterparties have with the Baseline Protocol Standard.

[R21] Testability: All requirements for [specification of the BPI APIs](#) are testable.

[R22]

Commercial Counterparties MUST publish their level of conformity (self-declaration or certification) with the Baseline Protocol Standard in a publicly accessible manner.

Publicly accessible in the context of this document means that there exists a URI or URL pointing to a publication specifying the level of conformity with this document that is accessible through the public internet.

[R22] Testability: Counterparties can publish their level of conformity as a W3C verifiable credential in a well-known location at the root level of their internet domain, public CCSM or IPFS.

2.7.2 CCSM Lifecycle Processes

Commercial Counterparties must comply with requirements of regulatory frameworks, e.g., Office of Foreign Assets Control ("OFAC") of the US Department of the Treasury when employing new operational and commercial frameworks as laid out in this standard. This means BPI participants and BPI operators must be able to provide compliance reports to authorities derived from a BPI that demonstrate compliance with the applicable regulatory rules such as Suspicious Activity Reports as required by the US Bank Secrecy Act of 1970.

[R23]

If required to meet particular third-party requirements, (e.g., privacy or regulatory frameworks in different jurisdictions), a commercial counterparty **MUST** record a pseudonymous map of the supply chain that is required to fulfill the provisioning of a request's commercial State Object (products, services or assets) transacted on a BPI.

Note, that in principle every product, service, or asset, or sets thereof has a supply chain. This requirement aims at situations where the product, service, or asset or sets thereof cannot be solely supplied by the Provider without sourcing components from other Providers.

Requesters are only aware of the identity and commercial data of their Provider(s), but not of the other participants in the supply chain. However, Requesters can cryptographically verify that a given set of claims by Providers about the supply chain are true, for example, that all supply chain participants are not located in an embargoed country.

Therefore, a pseudonymous map of a supply chain is a cryptographically connected and verifiable list of proofs about the relationships of participants and integrity of supply chain events that does not disclose identifying details of Providers and their commercial data.

This allows enforcement of conformance with regulations, additional legal and technical requirements without disclosure of confidential information.

An example of a pseudonymous map is given in the figure below:

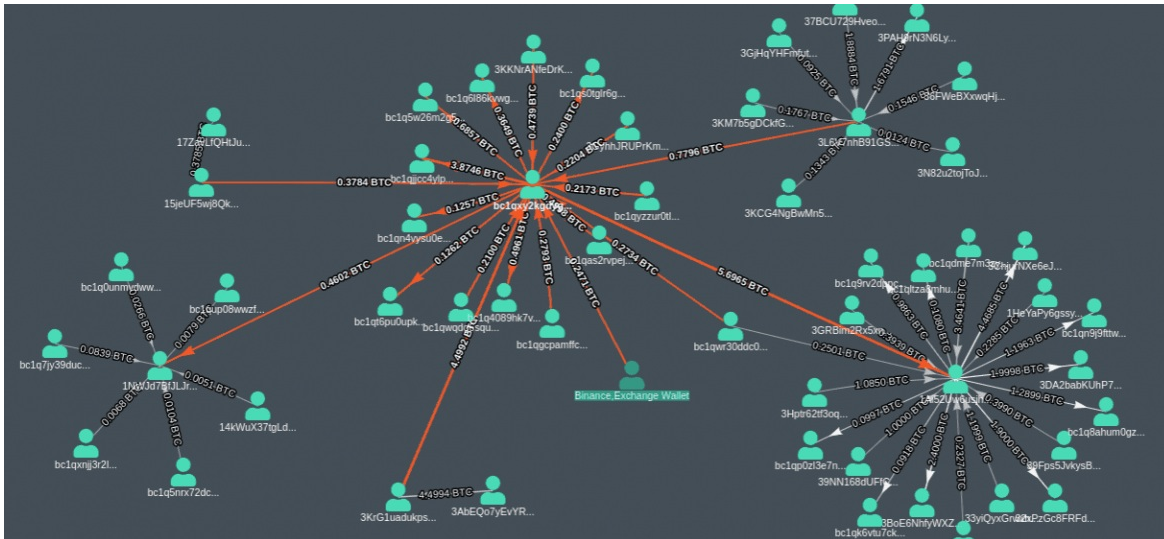


Figure 0: Illustrative example of a pseudonymous map using the example of Bitcoin transactions. Source: [Bitquery](#)

In the context of a supply chain in a BPI, the bitcoin values could be replaced by recursive zero-knowledge proofs. The map allows the BPI to validate the conformity of all the inputs and business rules of the supply chain that goes into a BPI State Object with a single proof.

[R23] Testability: A pseudonymous map, such as the example above, is naturally created by recursively proving the validity of a chain of commercial transactions using recursive zero-knowledge proof schemes such as PLONK used in the [Aztec Barretenberg Library](#) and its tests.

2.8 Baseline Protocol Reference Architecture

This section describes the components of the Baseline Reference Architecture:

- State Synchronization

Non-Standards Track Work Product

- CCSMs and BPI/CCSM Abstraction Layers
- External Applications
- Baseline Protocol Stack Detailed Reference Architecture Layers and Components

2.8.1 State Synchronization

A BPI can be used as a common frame of reference for business processes that can be used in a complementary way to existing System-of-Record integrations.

Illustrative High-Level Example

A Master Services Agreement (MSA) between a Requester (Buyer) and a Provider (Seller) is implemented on a BPI and contains billing terms, pricing, discounts, and Seller information such as billing address, etc. Once established and agreed upon by Buyer and Seller, the BPI provides state synchronization between Buyer and Seller since the ERP systems for Buyer and Seller can now refer to mutually agreed-upon data as a common frame of reference. Based on this mutually agreed-upon state in the MSA, the Buyer creates an Order in the business workflow based on the MSA and a cryptographic proof (in zero-knowledge) that confirms not only the correct application of business logic but also the correct application of commercial data in the Order creation. This proof is submitted together with the Order through the BPI and then validated by the Seller without having to utilize its System of Record for validation using the BPI. If the proof is validated, the Seller accepts the proposed state change by generating its cryptographic proof confirming its acceptance of the state change. The Seller then updates the state of the business workflow in the BPI and sends the new proof to the Buyer.

The figure below visually demonstrates high-level Buyer and Seller Order generation and acceptance assuming that an MSA between Buyer and Seller already exists and is recorded on a BPI and that the commercial state has been synchronized up to this workstep in the commercial business workflow.

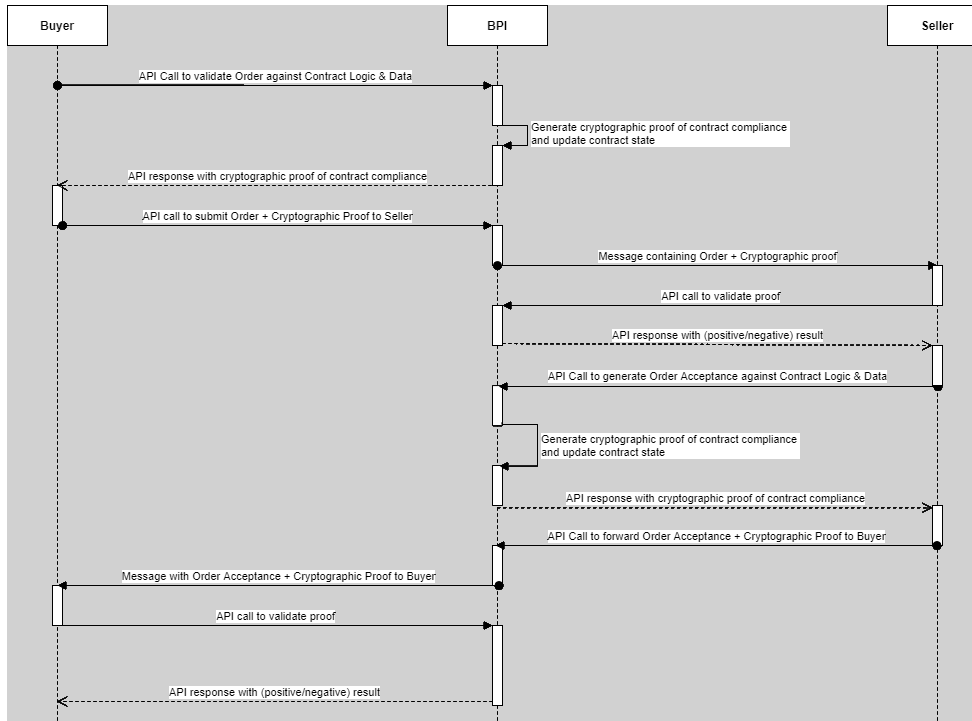


Figure 1: Schematic diagram illustrating how the commercial state between Buyer and Seller is synchronized and an Order created.

Without a BPI, both Buyer and Seller must assume that the MSA between them and all its values are correctly represented in the other party's respective Systems-of-Record. If an order is created based upon the MSA but does not comply with the MSA, it will likely result in extensive manual interactions between Seller and Buyer at one stage or another to resolve the problem to their mutual satisfaction.

[R24]

The transacting counterparties MUST agree on the business process rules which are represented in the business workflows and worksteps in the BPI.

[R24] Testability: A BPI can ensure both counterparties agree on business rules and business data required to validate the agreement by, for example, utilizing [payment term agreement](#) between two parties.

[R25]

The transacting counterparties MUST validate the correctness of a State Object based on a state change against the transaction business logic in the applicable BPI workflow and workstep.

[R25] Testability: A BPI can validate the correctness of a State Object based on a state change against transaction logic by verifying a zero-knowledge proof of the state change in the BPI, and ultimately on the utilized CCSM. Reference back to [\[R16\]](#) to understand the verification of a zero-knowledge proof in a smart contract.

[R26]

The transacting counterparties MUST generate a Proof of Correctness of a State Object based on a state change that can be validated against the BPI transaction business logic.

[R26] Testability: This can be accomplished by creating a [privacy package](#) with Zero-Knowledge Circuits generating Zero-Knowledge Proofs of the State changes of a State Object.

[R27]

Any new state between counterparties MUST be recorded on the BPI between them.

[R27] Testability: This can be accomplished by creating a new entry in the storage of the BPI for the new State, such as in this [example](#), which can be queried.

[R28]

Any transacting counterparty having received a Proof of Correctness of a state change MUST be able to validate that Proof of Correctness against the BPI between the counterparties.

[R28] Testability: Including a zero-knowledge proof, public input, and state commitment, together with the prover scheme, and if required the common reference string, inside the message payload

Non-Standards Track Work Product

would enable counterparties to validate the Proof of Correctness on their own utilizing known libraries for the prover scheme such a PLONK or Groth16.

[R29]

A transacting counterparty MUST include a Proof of Correctness of the State Object generated by the state change in the BPI Messages between the transacting counterparties.

[R29] Testability: A Proof of Correctness of the State Object can be stored and shared inside a Merkle tree, such as in this [example](#).

2.8.2 Considerations on BPI and CCSM Abstraction Layers and the CCSM Layer

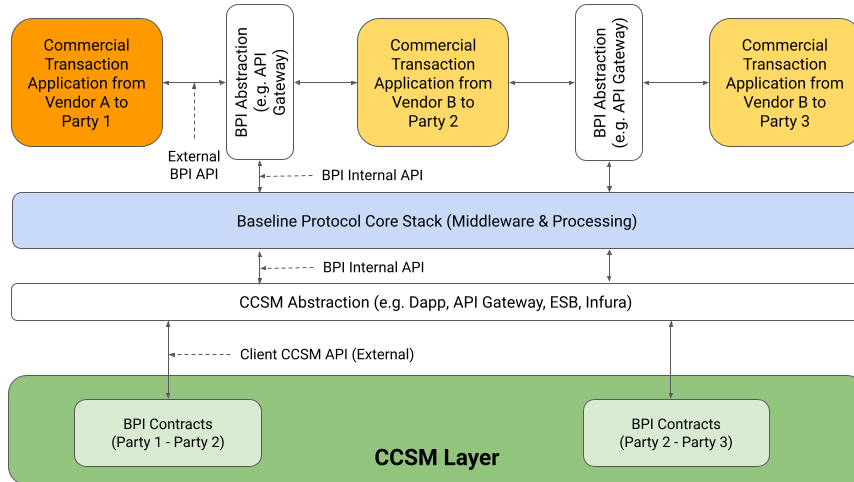


Figure 2: CCSM and BPI Abstraction Architecture

To maintain modularity in the reference architecture, this document introduces the concept of CCSM and BPI Abstraction. A CCSM or BPI Abstraction expressed through a BPI or CCSM Abstraction Layer, constitutes technology applications that wrap capabilities of CCSMs and BPIs such that these capabilities can be exposed to applications above the CCSM or BPI Abstraction Layers in a manner that minimizes the dependency of these applications on the details of a CCSM and BPI – Figure 2.

The Client CCSM API as an external BPI API is implementation-specific and will not be discussed further.

[R30]

CCSMs used in the implementation of a BPI MUST support bilateral and multi-lateral digital representations of contracts as defined in [Section 2.4.1](#).

[R30] Testability: Testability: All requirements in [Section 2.4.1](#) are testable.

[D8]

A CCSM or BPI Abstraction Layer used in a BPI SHOULD support more than one CCSM instance.

[D8] Testability: Support for this can come from the utilization of an adapter, such as in the example code for a simple Ethereum Adapter given in [\[R11\]](#).

[D9]

A CCSM or BPI Abstraction Layer used in a BPI SHOULD support more than one CCSM type.

This approach avoids lengthy discussions about which CCSM protocol to utilize for a BPI, simplifying the decision-making process considerably if most common CCSMs are incorporated.

Note that irrespective of whether one is in a public or private CCSM scenario, the protocol settings such as block time, consensus model, type of execution framework, etc. needs to be agreed upon by operating entities in some fashion either informally such as in [Ethereum](#) or formally such as in the Trade Finance consortium [Komgo](#).

The agreement on the governance entity, its rules, and its method of achieving interval synchronization consensus, as well as the definition of acceptable governance structures and their rules is beyond the scope of this document.

[D9] Testability: This can be achieved through the implementation of a new adapter program for each CCSM to add a CCSM into a BPI. Each adapter program will be different for each CCSM though all adapter programs will have a constructTx and a sendTransaction function, which was given in the example in [\[R11\]](#).

[R31]

The transacting counterparties MUST agree on which BPI is to be used.

[R31] Testability: In order for counterparties to transact on the same BPI instance, this requirement must have been fulfilled.

2.8.3 External Applications

[R32]

Application/s providing transaction functionality such as billing to counterparties, and are, therefore, external concerning the BPI, MUST be independent of any BPI.

Note, this requirement is motivated by reducing the dependency of counterparty internal systems on the BPI and vice versa.

[R32] Testability: In an implementation, counterparties only interact with the BPI utilizing the APIs as defined by the Baseline Protocol Standard, and not through the integration of functionalities of external systems directly into a BPI, for example integrating an API of an ERP system into a BPI.

2.8.4 Baseline Protocol Stack Detailed Reference Architecture Layers and Components

Non-Standards Track Work Product

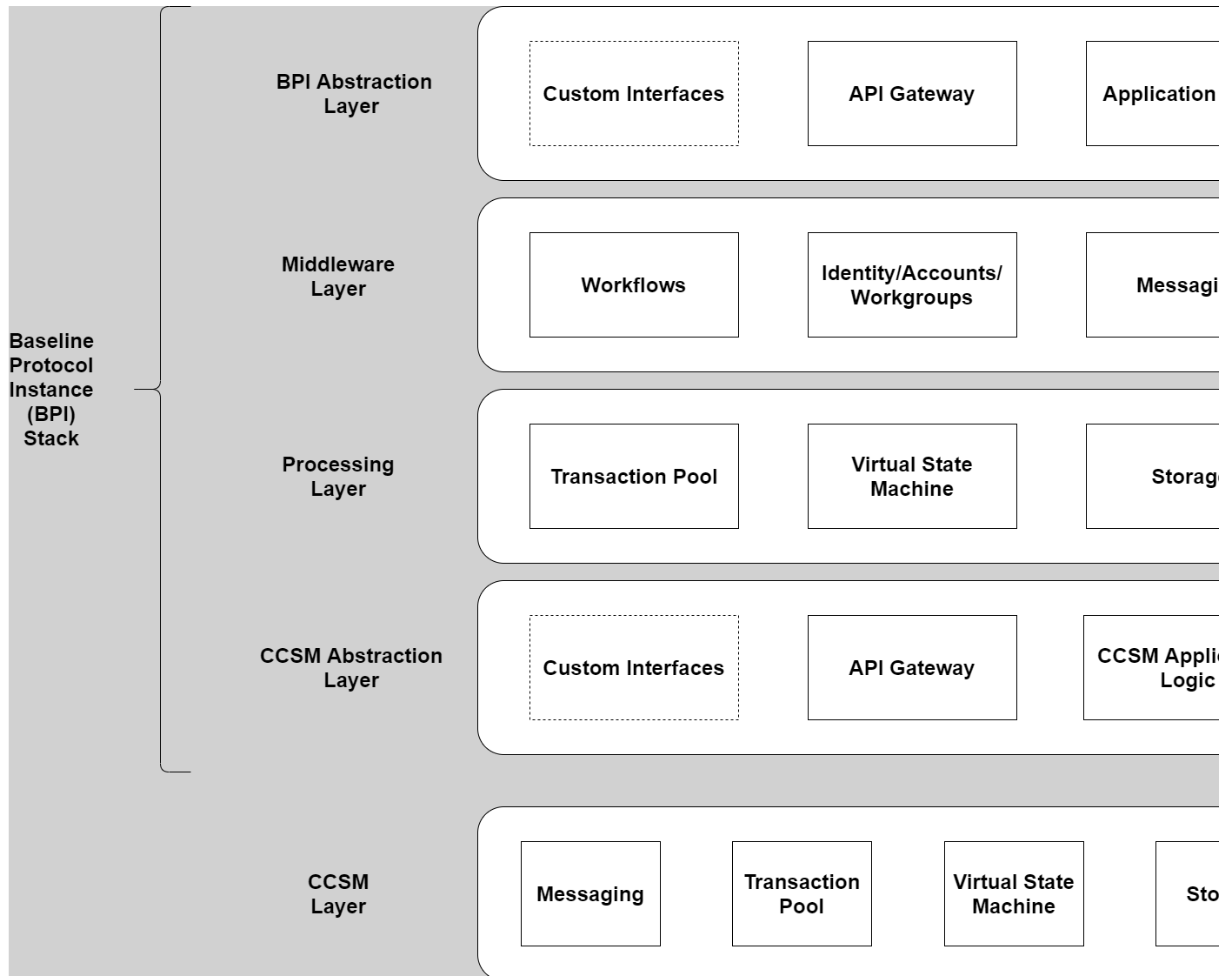


Figure 3: Detailed Baseline Reference Architecture Layers and Components

A Baseline Protocol Stack Reference Architecture as depicted above in Figure 3 is comprised of the following layers and in the following order:

- **Baseline Protocol (BPI) Abstraction Layer:** This layer enables accessing all externally available BPI functions through APIs as defined in the Baseline Protocol API Standards document
- **Middleware Layer:** This layer manages all counterparties to an agreement and its associated workflows and worksteps with business rules and business data as well as all counterparty delegates. In addition, it manages all messaging between counterparties to an agreement and instantiation of processing layers based on newly created or updated agreements and their workflows, worksteps, business rules, and business data.
- **Processing Layer:** Manages, properly sequences, and deterministically processes and finalizes in a privacy-preserving, cryptographically verifiable manner all state change requests from counterparties to agreements represented in the BPI.
- **CCSM Abstraction Layer:** This layer enables accessing all required BPI functions implemented on one or more CCSMs through APIs as defined in the Baseline Protocol API Standards document.
- **CCSM Layer:** This layer manages, properly sequences, and deterministically processes in a privacy-preserving, cryptographically verifiable manner all transactions from the Processing Layer as well as either deterministically or probabilistically finalizes on the CCSM all CCSM state transitions based on said transactions.

Below this document lists and defines the components of each layer as depicted in Figure 3. The detailed requirements for each component will be discussed in later sections of this document.

- **BPI Abstraction layer**
 - **API Gateway:** An API gateway that exposes all required functionality to the counterparties to an agreement and enforces all necessary authentication and authorization of API calls as well as properly directs the API calls within the Baseline Protocol Stack
 - **Application:** The application logic which manages the pre-processing and routing of all API requests, as well as the enforcement of authentication and authorization protocols and rules.
- **Middleware Layer**
 - **Workflows:** A Business Process Management engine that allows for the definition, management, and instantiation of workflows and worksteps and associated business rules and data based on agreements between counterparties
 - **Identity/Accounts/Workgroups:** A capability that allows for the identification and management of counterparties and their delegates as well as members of workflows and worksteps organized in workgroups that are derived from the counterparties to an agreement.
 - **Messaging:** A messaging capability that allows the exchange of secure and privacy-preserving messages between counterparties to an agreement to communicate and coordinate an agreement on proposed state changes.
- **Processing Layer**
 - **Transaction Pool:** one or more transaction pools that hold, properly sequence, preprocess and batch for processing by the Virtual State Machine all requested state change transactions of a BPI.
 - **Virtual State Machine:** one or more Virtual State Machines which deterministically process and finalize in a privacy-preserving, cryptographically verifiable manner all state change request transactions.
 - **Storage:** A storage system for the cryptographically linked current and historical state of all agreements in a BPI.

Non-Standards Track Work Product

- **CCSM Abstraction Layer**
 - **API Gateway:** An API gateway that enables accessing all required BPI functions implemented on one or more CCSMs, and properly directs the requests within the CCSM Abstraction layer to the proper CCSM API application logic.
 - **Application:** The CCSM API application logic manages the pre-processing, as well as the proper usage of the underlying CCSM and BPI authentication and authorization.
- **CCSM Layer**
 - **Messaging:** A messaging capability that allows the exchange of messages between CCSM nodes that comprise either received transactions or a new proposed CCSM state.
 - **Transaction Pool:** A transaction pool holds, properly sequences, pre-processes, and batches for processing by the CCSM Virtual State Machine all submitted CCSM transactions.
 - **Virtual State Machine:** A Virtual State Machine deterministically processes in a cryptographically verifiable manner all submitted transactions for CCSM state changes.
 - **Storage:** A storage system for the cryptographically linked current and historical state of all CCSM State Objects.

3 Identifiers, Identity and Credential Management

3.1 Introduction and High-Level Requirements

Currently, 3rd parties such as [Domain Name Services \(DNS\) registrars](#), Internet Corporation for Assigned Names and Numbers [\[ICANN\]](#), X.509 Certificate Authorities (see [\[X.509\]](#) and [\[CA\]](#)), or social media companies are responsible for the creation and management of online identifiers and the secure communication between them.

As evidenced over the last 20+ years, this design has demonstrated serious usability and security shortcomings.

When DNS and X.509 Public Key Infrastructure (PKIX) [\[NIST SP 800-32\]](#) was designed, the internet did not have a way to reliably agree upon the state of a registry (or database) with no trust assumptions. Consequently, standard bodies designated trusted 3rd parties (TTP) to manage identifiers and public keys. Today, virtually all Internet software relies on these authorities. These trusted 3rd parties, however, are central points of failure, where each is capable of compromising the integrity and security of large portions of the Internet. Therefore, once a TTP has been compromised, the usability of the identifiers it manages is also compromised.

As a result, companies spend significant resources fighting security breaches caused by CAs, and public internet communications that are both truly secure and user-friendly are still out of reach for most.

Given the above, the Baseline Protocol Standard identity approach is as follows: Every identity is controlled by its Principal Owner and not by a 3rd party unless the Principal Owner has delegated control to a 3rd party.

A Principal Owner is defined as the entity controlling the public key(s) which control the identity and its identifiers.

The Baseline Protocol Standard defines identity in the context of this document to mean the following:

Identity = <Identifier(s)> + <associated data>

where associated data refers to data describing the characteristics of the identity that is associated with the identifier(s). An example of such associated data could be an X.509 issued by a CA.

This approach requires a decentralized, or at least strongly federated, infrastructure as expressed in the requirements below.

[\[D10\]](#)

The Public Key Infrastructure (PKI) of a BPI SHOULD have no single point of failure, and SHOULD NOT require pre-existing trust relationships between participants.*

[\[D10\]](#) Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- All necessary participants and components are set up and operational within the BPI.
- The PKI component of the BPI is in place.
- No pre-existing trust relationships exist between participants.

Test Steps:

1. Simulate a failure scenario by shutting down one of the PKI nodes.
2. Disable all but one PKI node, leaving a single node operational.
3. Attempt to establish communication between two BPI participants who have not previously interacted. Initiate a communication between two participants who do not have any pre-existing trust relationship.

Expected Results:

1. The BPI PKI should still be operational without any significant disruptions or critical failures, demonstrating the absence of a single point of failure.
2. The BPI should successfully establish a secure communication channel between the participants without requiring pre-existing trust relationships.

[\[R33\]](#)

The PKI of a BPI MUST be strongly federated.

Strongly federated in this context means that there is a known, finite number of participants, without a single point of failure in the PKI. However, collusion of a limited number of participants in the federated infrastructure may still lead to a compromised PKI. The consensus thresholds required for a change in the infrastructure are out of scope for this document.

[\[R33\]](#) Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- All necessary participants and components are set up and operational within the BPI.
- The PKI component of the BPI is in place.
- A known, finite number of participants is registered with the PKI.
- There is no single point of failure in the PKI.

Test Steps:

1. Query the PKI for the list of registered participants.
2. Have a predefined set of participants attempt to collude and compromise the PKI.
3. Review the PKI architecture and configuration to ensure there is no single point of failure.

Expected Results:

1. The list should contain a known and finite number of participants, confirming that the PKI meets the requirement.
2. The PKI should detect and prevent the collusion, maintaining its integrity and security.
3. The PKI architecture should demonstrate resilience against single points of failure.

[\[R34\]](#)

The identifiers and identity utilized in a BPI MUST be controlled by its Principal Owner.

Non-Standards Track Work Product

For a BPI to properly operate, communication must be trusted and secure. Communications are secured through the safe delivery of public keys tied to identities. The Principal Owner of the identity uses a corresponding secret private key to both decrypt messages sent to them, and to prove they sent a message by signing it with its private key.

PKI systems are responsible for the secure delivery of public keys. However, the commonly used X.509 PKI (PKIX) undermines both the creation and the secure delivery of these keys.

In PKIX services are secured through the creation of keys signed by CAs. However, the complexity of generating and managing keys and certificates in PKIX has caused companies to manage the creation and signing of these keys themselves, rather than leaving them to their clients. This creates major security concerns from the outset, as it results in the accumulation of private keys at a central point of failure, making it possible for anyone with access to that repository of keys to compromise the security of connections in a virtually undetectable way.

The design of X.509 PKIX also permits any of the thousands of CAs to impersonate any website or web service. Therefore, entities cannot be certain that their communications are not being compromised by a fraudulent certificate allowing a PITM (Person-in-the-Middle) attack. While workarounds have been proposed, good ones do not exist.

Decentralized Public Key Infrastructure (DPKI) has been proposed as a secure alternative. The goal of DPKI is to ensure that, unlike PKIX, no single third-party can compromise the integrity and security of a system employing DPKI as a whole.

Within DPKI, a Principal Owner can be given direct control and ownership of a globally readable identifier by registering the identifier for example in a CCSM. Simultaneously, CCSMs allow for the assignment of arbitrary data such as public keys to these identifiers and permit those values to be globally readable in a secure manner that is not vulnerable to the PITM attacks that are possible in PKIX. This is done by linking an identifier's lookup value to the latest and most correct public keys for that identifier. In this design, control over the identifier is returned to the Principal Owner. Therefore, it is no longer trivial for any one entity to undermine the security of the entire DPKI system or to compromise an identifier that is not theirs, thus, overcoming the challenges of typical PKI.

Furthermore, DPKI requires a public registry of identifiers and their associated public keys that can be read by anyone but cannot be compromised. As long as this registration remains valid, and the Principal Owner can maintain control of their private key, no 3rd party can take ownership of that identifier without resorting to direct coercion of the Principal Owner.

[R34] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- The Decentralized Public Key Infrastructure (DPKI) is integrated into the BPI.
- A Principal Owner is identified and has control over their identity within the DPKI.
- A public registry of identifiers and their associated public keys is in place and secure.
- All necessary components for secure communication are operational.

Test Steps:

1. Access the DPKI registry and verify that the Principal Owner has control and ownership of their globally readable identifier.
2. Attempt to access and modify the public registry data to verify that it can be read by anyone but cannot be compromised.
3. Attempt to intercept and manipulate the communication between two parties within the BPI to verify that it is not vulnerable to PITM attacks.

Expected Results:

1. The Principal Owner should have direct control and ownership of their identifier within the DPKI.
2. The public registry should be securely readable by anyone, but attempts to compromise or modify it should fail.
3. The BPI communication should remain secure and resistant to PITM attacks.

[D11]

A BPI SHOULD utilize a DPKI.

[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- The Decentralized Public Key Infrastructure (DPKI) component is integrated into the BPI or available for integration.
- All necessary components for secure communication are operational.

Test Steps:

1. Examine the BPI configuration to determine if DPKI is present and properly configured.
2. Initiate a sample transaction or communication within the BPI and inspect the methods used for key management and identity verification.
3. Review the DPKI documentation and standards.

Expected Results:

1. The DPKI component should be integrated into the BPI or available for seamless integration.
2. The BPI should utilize the DPKI for key management and identity verification.
3. The BPI's use of DPKI should align with the principles and specifications defined for DPKI.

[CR1]>[D11]

Any Principal Owner in a DPKI system utilized by a BPI MUST be able to broadcast a message if it is well-formed within the context of the DPKI.

Other peers in the system do not require admission control. This implies a decentralized consensus mechanism naturally leading to the utilization of systems such as CCSMs.

[CR1]>[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- The Decentralized Public Key Infrastructure (DPKI) is integrated into the BPI or available for integration.
- A Principal Owner exists within the DPKI system.
- The DPKI system is decentralized and utilizes systems such as CCSMs for consensus.

Test Steps:

1. The Principal Owner initiates the process of creating and broadcasting a well-formed message within the context of the DPKI.
2. Check if other peers within the DPKI system have received the broadcasted message from the Principal Owner.
3. Evaluate the architecture and consensus process used within the DPKI system to ensure it is decentralized.

Expected Results:

1. The Principal Owner should be able to successfully create and broadcast a well-formed message without requiring admission control from other peers.
2. Expected Result: Other peers should be able to receive the broadcasted message without requiring admission control.
3. Expected Result: The DPKI system should employ a decentralized consensus mechanism, such as the use of systems like CCSMs.

[CR2]>[D11]

Non-Standards Track Work Product

Given two or more histories of DPKI updates, any Principal Owner within a BPI MUST be able to determine which one is preferred due to security by inspection.

This implies the existence of a method of ascertaining the level of resources backing a DPKI history such as the hash power in Bitcoin based on difficulty level and nonce.

Requirements of Identifier registration in DPKI are handled differently from DNS. Although registrars may exist in DPKI, these registrars must adhere to several requirements that ensure that identities belong to the entities they represent. This is achieved the following way:

[\[CR2\]>\[D11\]](#) Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- There are two or more histories of Decentralized Public Key Infrastructure (DPKI) updates available within the BPI.
- Principal Owners exist within the DPKI system.
- A method of ascertaining the level of resources backing a DPKI history, such as hash power, is available.
- Registrars may exist within DPKI but must adhere to the requirements ensuring that identities belong to the entities they represent.

Test Steps:

1. Check the BPI configuration to ensure the availability of multiple DPKI histories.
2. The Principal Owner inspects the available histories and selects the preferred one based on security criteria.
3. Evaluate the method and data used to calculate the level of resources backing a DPKI history, such as hash power in Bitcoin.
4. Review the requirements that registrars must adhere to in order to ensure that identities belong to the entities they represent.

Expected Results:

1. There should be two or more histories of DPKI updates accessible.
2. The Principal Owner should be able to determine the preferred history based on security criteria, such as resource backing or other relevant factors.
3. The method for ascertaining the resource level should be validated and reliable.
4. Registrars within the DPKI system should adhere to the specified requirements to confirm identity ownership.

[\[CR3\]>\[D11\]](#)

Private keys utilized in a BPI MUST be generated in a manner that ensures they remain under the Principal Owner's control.

[\[CR3\]>\[D11\]](#) Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Principal Owners are registered within the BPI.
- Private keys are generated and utilized within the BPI.
- The private key generation process is under scrutiny for compliance.

Test Steps:

1. Review the BPI's private key generation method and associated controls.
2. The Principal Owner initiates the process of generating a private key and demonstrates control over it.
3. Evaluate the BPI's adherence to established security standards and best practices for private key generation.

Expected Results:

1. The Principal Owner should be able to successfully generate a private key and prove that they have control over it.
2. The BPI's private key generation process should comply with recognized security standards and best practices.

[\[CR4\]>\[D11\]](#)

Generating key pairs in a BPI on behalf of the Principal Owner MUST NOT be allowed.

[\[CR4\]>\[D11\]](#) Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Principal Owners are registered within the BPI.
- The BPI includes functionality for generating key pairs.

Test Steps:

1. The Principal Owner initiates the process of generating a key pair.
2. Try to initiate key pair generation without the Principal Owner's direct action.

Expected Results:

1. The BPI should allow the Principal Owner to generate a key pair.
2. The BPI should not allow key pair generation on behalf of the Principal Owner without their direct involvement.

[\[CR5\]>\[D11\]](#)

Principal Owners in a BPI MUST always be in control of their identifiers and the corresponding public keys.

[\[CR5\]>\[D11\]](#) Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Principal Owners are registered within the BPI.
- Identifiers and corresponding public keys are managed within the BPI.
- Secure mechanisms for identifier and public key control are in place.

Test Steps:

1. The Principal Owner initiates a process to demonstrate control over their identifier.
2. The Principal Owner initiates a process to demonstrate control over their public key within the BPI.
3. An unauthorized party attempts to modify the identifier or public key of a Principal Owner.

Expected Results:

Non-Standards Track Work Product

1. The Principal Owner should be able to successfully validate their control over their identifier.
2. The Principal Owner should be able to successfully validate their control over their corresponding public key.
3. The BPI should not allow unauthorized parties to modify the identifier or public key of a Principal Owner.

[O1]

Principal Owners MAY extend control of their identifier to third parties.

For example for recovery purposes.

[O1] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- The Decentralized Public Key Infrastructure (DPKI) is integrated into the BPI or available for integration.
- A Principal Owner has control over their identifier within the DPKI.
- A third party is identified for potential extension of control.

Test Steps:

1. Access the DPKI registry and verify that the Principal Owner has control and ownership of their globally readable identifier.
2. Attempt to extend control of the Principal Owner's identifier to a third party as per the requirement.
3. Access the DPKI registry and inspect the ownership and control of the extended identifier by the third party.

Expected Results:

1. The Principal Owner should have direct control and ownership of their identifier within the DPKI.
2. The BPI should support the extension of control of the identifier to a third party, allowing the Principal Owner to grant access and control to the designated third party.
3. The third party should have direct control and ownership of the extended identifier within the DPKI.

[CR6]<[O1]

Extension of control of identifiers to 3rd parties in a BPI MUST be an explicit, informed decision by the Principal Owner of such identifiers.

[CR6]>[O1] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Principal Owners are registered within the BPI.
- The BPI includes functionality for controlling identifiers.
- A third party is identified for potential extension of control.

Test Steps:

1. Attempt to extend control of an identifier to a third party without the explicit consent of the Principal Owner.
2. Communicate with the Principal Owner to receive their consent for the extension of control to the identified third party.
3. Complete the process of extending control to the third party, ensuring the Principal Owner's consent is honored.

Expected Results:

1. The BPI should not allow the extension of control to a third party without the explicit and informed consent of the Principal Owner.
2. The Principal Owner should explicitly and willingly consent to the extension of control to the third party.
3. The BPI should allow the extension of control to the third party as an explicit, informed decision by the Principal Owner.

[R35]

Private keys MUST be stored and/or transmitted securely.

No mechanism should exist that would allow a single entity to deprive a Principal Owner of their identifier without their consent. This implies that:

[R35] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Private keys are generated and used within the BPI.
- Secure storage and transmission mechanisms are in place for private keys.

Test Steps:

1. Review the BPI configuration and settings related to private key storage.
2. Initiate a secure transmission of a private key. Attempt to intercept and compromise the transmission of the private key. Verify the security of the transmission.
3. Review the security standards and best practices followed for private key storage and transmission. Evaluate the BPI's adherence to these established security practices.

Expected Results:

1. Private keys should be stored in a secure manner that is not easily accessible to unauthorized users.
2. The private key transmission should be securely encrypted and resistant to interception or compromise.
3. The BPI's handling of private keys should comply with recognized security standards and best practices.

[CR7]<[D11]

Once a namespace is created within the context of a DPKI, it MUST NOT be possible to destroy it.

[CR7]<[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A namespace has been created within the context of a Decentralized Public Key Infrastructure (DPKI) using the BPI.
- Secure mechanisms for namespace management are in place.

Test Steps:

1. Attempt to destroy the previously created namespace.
2. Check if the previously created namespace still exists and is accessible.

Non-Standards Track Work Product

Expected Results:

1. The BPI should not allow the destruction of the namespace.
2. The namespace should be preserved and accessible, and it should not have been destroyed.

[CR8]<[D11]

Namespaces in a DPKI utilized by a BPI MUST NOT contain blacklisting mechanisms that would allow anyone to invalidate identifiers that do not belong to them.

[CR8]<[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A namespace exists within the context of a Decentralized Public Key Infrastructure (DPKI) using the BPI.
- Secure mechanisms for namespace management are in place.

Test Steps:

1. Review the configuration and settings of the namespace within the DPKI to check for the presence of blacklisting mechanisms.
2. Attempt to use the namespace or associated functionality to invalidate an identifier that does not belong to the user.

Expected Results:

1. The namespace should not contain any blacklisting mechanisms that would allow anyone to invalidate identifiers that do not belong to them.
2. Expected Result: The BPI should not allow the invalidation of identifiers that do not belong to the user, confirming the absence of blacklisting mechanisms.

[CR9]<[D11]

The rules for registering and renewing identifiers in a DPKI utilized by a BPI MUST be transparent and expressed in simple terms.

[CR9]<[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A Decentralized Public Key Infrastructure (DPKI) is integrated into the BPI or available for integration.
- Rules for registering and renewing identifiers within the DPKI are established.

Test Steps:

1. Review the documentation or configuration that defines the rules for identifier registration.
2. Review the documentation or configuration that defines the rules for identifier renewal.

Expected Results:

1. The rules for registering identifiers should be transparent and expressed in simple terms that are easily understandable.
2. The rules for renewing identifiers should be transparent and expressed in simple terms that are easily understandable.

[R36]

If registration is used as security to an expiration policy, the Principal Owner MUST be explicitly and timely warned that failure to renew the registration on time could result in the Principal Owner losing control of the identifier.

[R36] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Principal Owners are registered within the BPI.
- An expiration policy linked to registration exists within the BPI.
- The BPI includes mechanisms for sending warnings to Principal Owners.
- Timely warning thresholds are defined for registration expiration.

Test Steps:

1. Review the configuration and settings within the BPI to check for the existence of an expiration policy linked to registration.
2. Verify that the BPI specifies the time frame within which Principal Owners will be warned about impending registration expiration.
3. Review the documentation or configuration to understand how warnings are delivered to Principal Owners.
4. Trigger a scenario where a Principal Owner's registration is nearing expiration and observe the warning mechanism in action.

Expected Results:

1. The BPI should have a registration expiration policy in place.
2. Timely warning thresholds should be clearly defined.
3. The BPI should have mechanisms in place to send warnings to Principal Owners.
4. The Principal Owner should receive an explicit and timely warning about the impending expiration, explicitly stating that failure to renew on time could result in loss of control of the identifier.

[CR10]>[D11]

Once set, namespace rules within a DPKI utilized by a BPI MUST NOT be altered to introduce any new restrictions for renewing or updating identifiers.

Otherwise, it would be possible to take control of identifiers away from Principal Owners without their consent.

[CR10]>[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A namespace exists within the context of a Decentralized Public Key Infrastructure (DPKI) using the BPI.
- Namespace rules are established for renewing or updating identifiers within the DPKI.
- Secure mechanisms for managing namespace rules are in place.

Test Steps:

1. Review the documentation or configuration that defines the current namespace rules.
2. Attempt to alter the existing namespace rules to introduce new restrictions for renewing or updating identifiers.

Expected Results:

Non-Standards Track Work Product

1. The existing namespace rules should be documented and stable.
2. The BPI should not allow the alteration of namespace rules to introduce new restrictions for renewing or updating identifiers.

[CR11]>[D11]

Within a DPKI utilized by a BPI, processes for renewing or updating identifiers MUST NOT be modified to introduce new restrictions for updating or renewing an identifier, once issued.

[Cr11]>[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A Decentralized Public Key Infrastructure (DPKI) is integrated into the BPI or available for integration.
- Processes for renewing or updating identifiers within the DPKI are established.
- Secure mechanisms for managing identifier renewal and updates are in place.

Test Steps:

1. Examine the existing processes for renewing or updating identifiers within the DPKI.
2. Attempt to modify the existing processes for identifier renewal and updates to introduce new restrictions.

Expected Results:

1. The existing processes for renewing or updating identifiers should be documented and stable.
2. The BPI should not allow the modification of processes to introduce new restrictions for renewing or updating an identifier once it has been issued.

[CR12]>[D11]

Within a DPKI utilized by a BPI, all network communications for creating, updating, renewing, or deleting identifiers MUST be sent via a non-centralized mechanism.

This is necessary to ensure that a single entity cannot prevent identifiers from being updated or renewed.

[Cr12]>[D11] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A Decentralized Public Key Infrastructure (DPKI) is integrated into the BPI or available for integration.
- Processes for creating, updating, renewing, or deleting identifiers within the DPKI are established.
- Secure non-centralized network communication mechanisms are in place.

Test Steps:

1. Examine the network communication mechanisms used for creating, updating, renewing, or deleting identifiers within the DPKI.
2. Ensure that communications are not centralized through a single point or entity.

Expected Results:

1. All network communications related to identifier management should follow a non-centralized path, avoiding single points of failure.

3.2 BPI Identifiers, Identities and Credentials, and their Management

Building on the requirements in section [3.1 Introduction and High-Level Requirements](#), this section focuses on identifiers, identities, and credentials used within a BPI or a network of BPIs. Note that BPI interoperability - which will be discussed in section [5 Middleware, Communication and Interoperability](#) - is predicated on known, discoverable, and identifiable approaches to how identifiers and credentials are created, updated, revoked, and deleted and how standardized identity frameworks are related to those identifiers and credentials utilized in one or more BPIs.

In the following, this document will use Requester and Provider as established in this document to refer to the entities making and those receiving requests.

3.2.1 BPI Identifiers

Uniqueness and security of BPI identifiers are very important to unambiguously identify entities interacting with and through one or more BPIs and keep those interactions secure. Furthermore, to facilitate automation and real-time interactions within and through a BPI, the discovery of identifiers and an ability to resolve them to the underlying public keys that secure them is also critical.

[R37]

Requester and Provider interacting with and through a BPI, as well as any BPI Operator, MUST each have a unique identifier.

[R37] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Requesters, Providers, and BPI Operators are registered within the BPI.
- Secure mechanisms for identifier assignment are in place.

Test Steps:

1. Review the identifiers assigned to Requesters, Providers, and BPI Operators within the BPI to ensure that each entity has a unique identifier .
2. Attempt to assign the same identifier to multiple Requesters, Providers, or BPI Operators within the BPI.

Expected Results:

1. Each Requester, Provider, and BPI Operator should have a unique and distinct identifier assigned to them.
2. The BPI should not allow the assignment of the same identifier to multiple entities, ensuring uniqueness.

[R38]

Any unique identifier utilized within a BPI MUST be associated with a set of public keys.

[R38] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to entities within the BPI (e.g., Requesters, Providers, BPI Operators).
- Public keys are generated and available for association with identifiers.
- Secure mechanisms for associating public keys with identifiers are in place.

Test Steps:

Non-Standards Track Work Product

1. Review the identifiers assigned to entities within the BPI and check if each identifier is associated with a set of public keys.
2. Attempt to disassociate public keys from an identifier within the BPI.

Expected Results:

1. Each unique identifier within the BPI should be associated with the appropriate set of public keys.
2. The BPI should not allow the disassociation of public keys from an identifier, ensuring that all identifiers remain associated with their respective public keys.

[R39]

Any unique identifier utilized within a BPI MUST be discoverable by any 3rd party within said BPI.

[R39] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to entities within the BPI (e.g., Requesters, Providers, BPI Operators).
- Secure mechanisms for identifier discoverability are in place.

Test Steps:

1. Review the identifiers assigned to entities within the BPI and confirm that unique identifiers can be discovered by any 3rd party participant within the BPI.
2. Attempt to configure or test scenarios where unique identifiers cannot be discovered by any 3rd party within the BPI.

Expected Results:

1. Each unique identifier within the BPI should be discoverable by any 3rd party participant without any access restrictions.
2. The BPI should not allow configurations or scenarios that restrict or prevent the discoverability of unique identifiers by any 3rd party participant.

[R40]

Any unique identifier utilized within a BPI MUST be resolvable to its associated public keys used for cryptographic authentication of the unique identifier.

[R40] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to entities within the BPI (e.g., Requesters, Providers, BPI Operators).
- Each identifier is associated with its corresponding set of public keys used for cryptographic authentication.
- Secure mechanisms for identifier resolution are in place.

Test Steps:

1. Review the unique identifiers assigned to entities within the BPI and confirm that each identifier is associated with its corresponding set of public keys used for cryptographic authentication.
2. Attempt to configure or test scenarios where unique identifiers cannot be resolved to their associated public keys.

Expected Results:

1. Each unique identifier within the BPI should have an established association with its corresponding set of public keys.
2. The BPI should not allow configurations or scenarios that disrupt or prevent the resolution of unique identifiers to their associated public keys.

[R41]

Any unique identifier utilized within a BPI MUST be resolvable to an endpoint as a URI that identifies the Baseline Protocol Standard as a supported protocol including the supported version(s).

[R41] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to entities within the BPI (e.g., Requesters, Providers, BPI Operators).
- Each identifier is associated with an endpoint URI that identifies the Baseline Protocol Standard as a supported protocol, including supported version(s).
- Secure mechanisms for identifier resolution to URIs are in place.

Test Steps:

1. Inspect the records or configuration to ensure that each identifier can be resolved to a URI with the required protocol information.
2. Initiate processes or configurations that disrupt or prevent the resolution of identifiers to URIs with the required protocol information.

Expected Results:

1. Each unique identifier within the BPI should have an associated endpoint URI that correctly identifies the Baseline Protocol Standard as a supported protocol, including supported version(s).
2. The BPI should not allow configurations or scenarios that disrupt or prevent the resolution of unique identifiers to URIs with the required protocol information.

[R42]

Any unique identifier utilized within a BPI MUST be resolvable to an endpoint as a URI that allows for BPI messaging.

[R42] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to entities within the BPI (e.g., Requesters, Providers, BPI Operators).
- Each identifier is associated with an endpoint URI that allows for BPI messaging.
- Secure mechanisms for identifier resolution to BPI messaging URIs are in place.

Test Steps:

1. Review the unique identifiers assigned to entities within the BPI and confirm that each identifier is resolvable to an endpoint URI that allows for BPI messaging.
2. Attempt to configure or test scenarios where unique identifiers cannot be resolved to URIs that allow for BPI messaging.

Expected Results:

1. Each unique identifier within the BPI should have an associated endpoint URI that enables BPI messaging, ensuring that BPI-related communication can occur.
2. The BPI should not allow configurations or scenarios that disrupt or prevent the resolution of unique identifiers to URIs capable of facilitating BPI messaging.

Non-Standards Track Work Product

[D12]

Any unique identifier utilized within a BPI SHOULD follow the W3C DID Core specification [\[W3C DID\]](#).

[D12] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to entities within the BPI (e.g., Requesters, Providers, BPI Operators).
- There is a requirement to follow the W3C DID Core specification for these identifiers.
- Secure mechanisms for handling and validating W3C DID-compliant identifiers are in place.

Test Steps:

1. Review the unique identifiers assigned to entities within the BPI and confirm that they follow the format specified in the W3C DID Core specification.
2. Attempt to configure or use identifiers that do not comply with the W3C DID Core specification.

Expected Results:

1. Each unique identifier within the BPI should follow the format prescribed by the W3C DID Core specification.
2. The BPI should not allow configurations or scenarios that result in the use of identifiers that do not comply with the W3C DID Core specification.

3.2.2 BPI Identities and Credentials

After having discussed the minimal set of requirements on identifiers utilized in a BPI, it is important to discuss how these relate to identity and claims about facts relevant to Requester, Provider, and BPI Operator, also called credentials.

Before this document can discuss requirements it needs to establish the scope of identity and credential management within the context of a BPI.

In the figure below, this document establishes the context and scope of identity and credential management for a BPI.

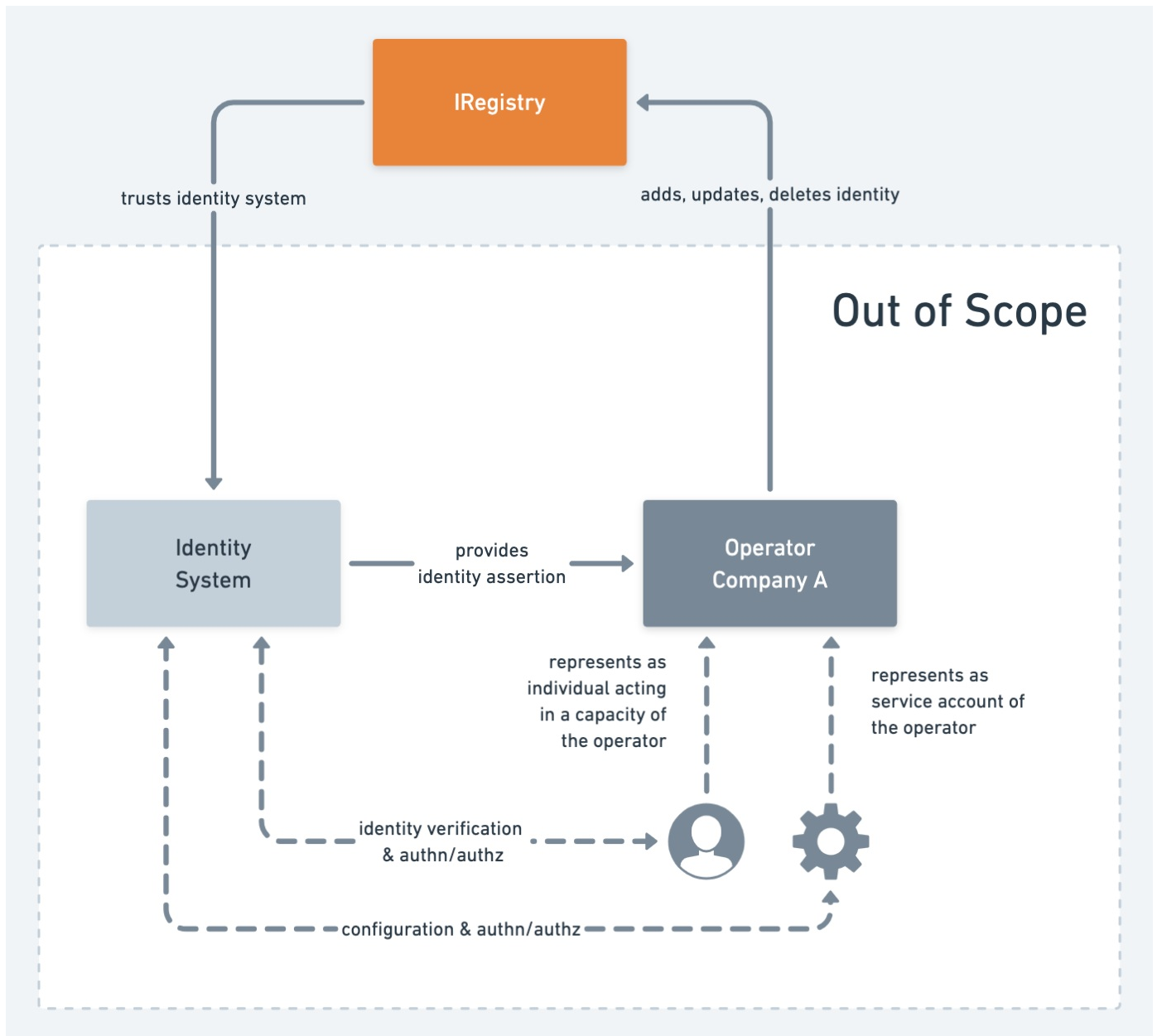


Figure 4: Schematic delineation of the Identity and Credential scope of a BPI; Source: Oliver Terbu (ConsenSys)

As depicted, identities and credentials are established outside of the context, and, therefore, the scope of a BPI. Hence, it is incumbent on BPI participants – Requesters, Providers, and, if distinct, Operators – to establish the trust context of acceptable identities and credentials for a BPI. This statement also applies to a network of BPIs which are to interoperate with one another.

[D13]

A unique identifier utilized within one or more BPIs SHOULD be linked to an entity accepted by BPI participants through a cryptographically signed, cryptographically verifiable, and cryptographically revocable credential based on the public keys associated with the unique identifier of the credential issuer.

An entity can be a Legal Entity where Legal Entity is an individual, organization, or company that has legal rights and obligations.

Note that credentials utilized within one or more BPIs may be self-issued. The acceptance of self-issued credentials is up to the BPI participants that need to rely on the claim(s) within a self-issued credential.

[D13] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to entities within the BPI (e.g., Requesters, Providers, BPI Operators).
- Cryptographically signed, verifiable, and revocable credentials based on public keys are supported within the BPI.
- Cryptographically signed, verifiable, and revocable credentials are issued and associated with unique identifiers.
- Entities, including self-issued credentials, are accepted by BPI participants based on these credentials.

Test Steps:

1. Check if cryptographically signed, verifiable, and revocable credentials exist for the unique identifiers utilized within the BPI.

Non-Standards Track Work Product

2. Confirm that entities linked to unique identifiers are accepted by BPI participants based on the associated cryptographically signed credentials.
3. Attempt to revoke cryptographically signed credentials and confirm that BPI participants no longer accept the associated entities.

Expected Results:

1. Cryptographically signed credentials should be available for the specified unique identifiers.
2. Entities linked to unique identifiers through cryptographically signed credentials should be accepted by BPI participants for interactions.
3. Entities associated with revoked credentials should not be accepted by BPI participants for interactions.

[R43]

The unique identifier of the (Legal) Entity MUST be the subject of the credential.

[R34] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to (Legal) Entities within the BPI.
- Cryptographically signed credentials are issued and associated with unique identifiers.
- The credentials have a subject field that designates the unique identifier of the (Legal) Entity.
- Entities are accepted by BPI participants based on the associated credentials.

Test Steps:

1. Review the credential details and data to verify that the subject field accurately matches the unique identifier of the (Legal) Entity.
2. Attempt to validate credentials by verifying that the subject field matches the unique identifier of the (Legal) Entity.

Expected Results:

1. The subject field of each credential should correctly designate the unique identifier of the associated (Legal) Entity.
2. Validated credentials should pass the check, confirming that the subject field matches the unique identifier of the (Legal) Entity.

[R44]

The unique identifier of the issuer of the (Legal) Entity credential utilized in one or more BPIs MUST have a credential linking the unique identifier of the issuer to an (Legal) Entity accepted by the participants within aforementioned BPIs.

[R44] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Unique identifiers are assigned to (Legal) Entities within the BPI.
- Cryptographically signed credentials are issued and associated with unique identifiers.
- The credentials have a subject field designating the unique identifier of the (Legal) Entity.
- There is an issuer credential associated with the (Legal) Entity issuer's unique identifier.
- Entities are accepted by BPI participants based on the associated credentials.

Test Steps:

1. Review the BPI configuration and data to ensure that an issuer credential exists for the issuer's unique identifier.
2. Confirm that the (Legal) Entity issuer's unique identifier is accepted by BPI participants based on the issuer credential associated with it.
3. Attempt to revoke the issuer credential and confirm that BPI participants no longer accept the (Legal) Entity issuer's unique identifier.

Expected Results:

1. An issuer credential should be available for the specified issuer's unique identifier.
2. Entities linked to the (Legal) Entity issuer's unique identifier through the issuer credential should be accepted by BPI participants for interactions.
3. Entities associated with the revoked issuer credential should not be accepted by BPI participants for interactions.

[D14]

A credential utilized within one or more BPIs SHOULD follow the W3C Verifiable Credential Standard [\[W3C VC\]](#).

[D14] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Credentials are used within the BPI for entities, issuers, or other purposes.
- There is a requirement for these credentials to follow the W3C Verifiable Credential Standard.
- Cryptographically signed, verifiable, and revocable credentials are supported within the BPI.

Test Steps:

1. Review the credential details and data to verify that they comply with the requirements of the W3C Verifiable Credential Standard.
2. Attempt to configure or use credentials that do not comply with the W3C Verifiable Credential Standard.

Expected Results:

1. Credentials within the BPI should follow the format and standards prescribed by the W3C Verifiable Credential Standard.
2. The BPI should not allow configurations or scenarios that result in the use of credentials that do not comply with the W3C Verifiable Credential Standard.

[R45]

A credential utilized within one or more BPIs MUST itself have a unique and resolvable identifier.

Note, that the unique and resolvable identifier of a credential does not have to be associated with any cryptographic keys.

[R45] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Credentials are used within the BPI for entities, issuers, or other purposes.
- There is a requirement for these credentials to have a unique and resolvable identifier.
- Cryptographically signed, verifiable, and revocable credentials are supported within the BPI.

Non-Standards Track Work Product

Test Steps:

1. Review the credential details and data to verify that each credential has a unique and resolvable identifier.
2. Attempt to configure or use credentials that do not have a unique and resolvable identifier.

Expected Results:

1. Each credential within the BPI should have a unique and resolvable identifier.
2. The BPI should not allow configurations or scenarios that result in the use of credentials without a unique and resolvable identifier.

[R46]

If present, the status of a credential utilized within one or more BPIs MUST be discoverable by a party verifying the credential, the credential verifier.

In the context of this document, a credential verifier is defined per the W3C Verifiable Credential Standard [\[W3C VC\]](#).

[R46] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Credentials are used within the BPI for entities, issuers, or other purposes.
- There is a requirement for the status of these credentials to be discoverable by a party verifying the credential, the credential verifier.
- Cryptographically signed, verifiable, and revocable credentials are supported within the BPI.

Test Steps:

1. Review the credential details and data to verify that the status of each credential is discoverable by a credential verifier.
2. Attempt to configure or use credentials for which the status cannot be discovered by a party verifying the credential.

Expected Results:

1. The status of each credential within the BPI should be discoverable by a party verifying the credential, as per the requirements of the W3C Verifiable Credential Standard.
2. The BPI should not allow configurations or scenarios that result in the use of credentials for which the status cannot be discovered by a credential verifier, as required.

[D15]

A credential utilized within one or more BPIs SHOULD be discoverable by a participant in said BPI(s).

Credential discoverability in the context of this document means that a BPI Subject can discover credentials of other BPI Subjects utilized within a BPI or across BPIs, if relevant for BPI interoperability.

Note that discoverability can be restricted based on privacy and / or security rules within a given BPI. Discoverability could be achieved for example through a credential registry within a BPI or by listing a credential access endpoint in the DID document of a BPI Subject.

[D15] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Credentials are used within the BPI for entities, issuers, or other purposes.
- There is a requirement for credentials to be discoverable by participants within the BPI(s) as specified in the requirement.
- Cryptographically signed, verifiable, and revocable credentials are supported within the BPI.
- Privacy and security rules governing discoverability are defined within the BPI.

Test Steps:

1. Review the BPI configuration, credential details, and any relevant privacy and security rules to verify that credentials can be discovered by participants within the BPI(s).
2. Attempt to configure or use credentials in scenarios where their discoverability is restricted based on privacy and security rules within the BPI.

Expected Results:

1. The BPI should support discoverability of credentials by participants within the BPI(s) according to the specified privacy and security rules.
2. The BPI should enforce the privacy and security rules to restrict discoverability of credentials as per the configured scenarios.

[R47]

The presentation of a credential utilized within one or more BPIs MUST be cryptographically signed by the presenter of the credential, also known as the holder.

See the W3C Verifiable Credential Standard [\[W3C VC\]](#) for a definition of credential holder [\[Holder\]](#).

[R47] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- Credentials are used within the BPI for entities, issuers, or other purposes.
- There is a requirement that the presentation of credentials must be cryptographically signed by the presenter (holder).
- Cryptographically signed, verifiable, and revocable credentials are supported within the BPI.

Test Steps:

1. Perform a credential presentation process and examine the presented credential to verify that it is cryptographically signed by the presenter (holder).
2. Attempt to configure or use a credential presentation process where the presented credential is not cryptographically signed by the presenter (holder).

Expected Results:

1. The presented credential should include a valid cryptographic signature by the presenter (holder).
2. The BPI should not allow configurations or scenarios that result in the presentation of unsigned credentials.

[R48]

If a credential holder is a BPI participant, the holder MUST have a unique identifier that has been established within the context the holder operates in.

[R48] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.

Non-Standards Track Work Product

- A credential holder is a BPI participant.
- The BPI participant is operating within a specific context or environment.

Test Steps:

1. Examine the BPI participant's credentials or information to determine if a unique identifier is present.
2. Check for any duplicates or conflicting identifiers within the same context.

Expected Results:

1. The BPI participant should have a unique identifier established within their operating context.
2. The identifier should be unique and not conflict with identifiers of other participants in the same context.

As discussed in section [3.1 Introduction and High-Level Requirements](#), BPIs require either decentralized or strongly federated identifier/identity providers that have been agreed to by the participants in a BPI context of one or more BPIs.

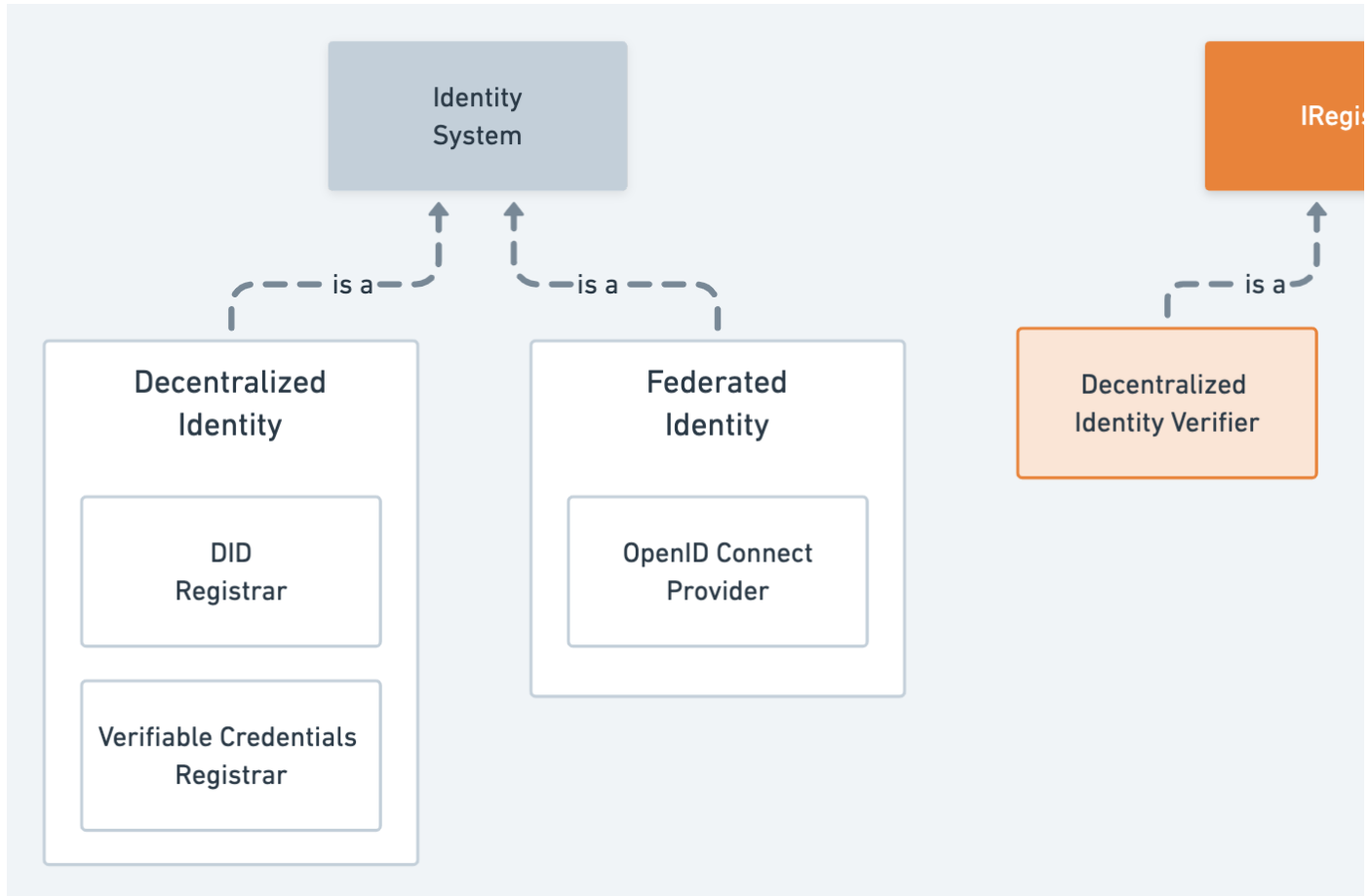


Figure 5: Example of a Delineation of the Identity and Credential issuing authorities used in a BPI and their management within a BPI using Decentralized Identity Verifiers and OpenID Connect Relying Party as examples; Source: Oliver Terbu (ConsenSys)

As depicted in Figure 5 above, the accepted Entity identity credentials, or other credentials from Identity providers, that are presented by a BPI participant need to be verified by the BPI against the issuing providers. Once validated, credentials are stored in the BPI.

OpenID Connect Identity Provider [OIDC] is an example for a federated identity provider and a DID or Verifiable Credentials Registry which is typically built using a CCSM as an example of a decentralized identity provider.

For a BPI to achieve these objectives, the following requirements need to be met:

[R49]

A unique identifier utilized in a BPI MUST be stored by the BPI.

[R49] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A unique identifier is generated or used within the BPI for entities, issuers, or other purposes.
- There is a requirement that the BPI must store the unique identifier.

Test Steps:

1. Generate or use a unique identifier within the BPI.
2. Examine the BPI's data storage components and records to verify that the unique identifier is stored by the BPI.
3. Query the BPI's data storage to retrieve the stored unique identifier.

Expected Results:

Non-Standards Track Work Product

1. The BPI should successfully generate or utilize the unique identifier.
2. The unique identifier should be present in the BPI's storage, confirming that it is stored as required.
3. The BPI should provide a valid response with the retrieved unique identifier, confirming that it is stored and retrievable.

[R50]

The Principal Owner or their delegates MUST prove control over a unique identifier utilized in a BPI every time said unique identifier is used in the BPI by the Principal Owner or their delegates.

[R50] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A unique identifier is generated or used within the BPI for entities, issuers, or other purposes.
- There is a requirement that the Principal Owner or their delegates must prove control over the unique identifier every time it is used.

Test Steps:

1. Create or utilize a unique identifier within the BPI.
2. Initiate an action or transaction within the BPI that involves the use of the unique identifier by the Principal Owner or their delegates.
3. Test Action: Repeat Step 2 for multiple actions or transactions within the BPI, ensuring that control over the unique identifier is verified every time it is used.

Expected Results:

1. The BPI should successfully generate or utilize the unique identifier.
2. The Principal Owner or their delegates should be able to prove control over the unique identifier as required.
3. The Principal Owner or their delegates should be able to consistently prove control over the unique identifier for each action or transaction.

[R51]

Every time a unique identifier utilized in a BPI is used in the BPI by the Principal Owner or their delegates, the BPI MUST verify that the Principal Owner or their delegates are in control of said unique identifier.

Note that proof of control might be performed by a relying party if authority has been delegated.

In the context of this document, a relying party is defined per the W3C Verifiable Credential Standard [W3C VC].

[R51] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A unique identifier is generated or used within the BPI for entities, issuers, or other purposes.
- There is a requirement that every time the unique identifier is used by the Principal Owner or their delegates in the BPI, the BPI must verify their control over it.
- Authority delegation mechanisms are in place, allowing relying parties to perform control verification when authority has been delegated.

Test Steps:

1. Generate or use a unique identifier within the BPI.
2. Perform an action or transaction within the BPI involving the unique identifier.
3. Perform additional actions or transactions within the BPI involving the unique identifier.

Expected Results:

1. The BPI should successfully generate or utilize the unique identifier.
2. The BPI or relying party (if delegated) should be able to confirm control over the unique identifier as required.
3. The BPI or relying party (if delegated) should be able to consistently confirm control over the unique identifier for each action or transaction.

[D16]

A credential utilized in a BPI SHOULD be stored in the BPI.

This avoids the re-presentation of the credential after the initial presentation as long as those credentials are valid.

[D16] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A credential is generated or utilized within the BPI for entities, issuers, or other purposes.
- There is a requirement that credentials utilized in the BPI should be stored within the BPI.

Test Steps:

1. Generate or use a credential within the BPI.
2. Examine the BPI's data storage components and records.
3. Attempt to re-present the previously used credential within the BPI for an action or transaction that requires it.

Expected Results:

1. The BPI should successfully generate or utilize the credential.
2. The credential should be present in the BPI's storage, confirming that it is stored as required.
3. The BPI should accept and validate the stored credential for re-presentation, avoiding the need for re-entering or re-generating the credential.

[R52]

A credential holder MUST prove control over a credential utilized in a BPI every time said credential is presented to the BPI or a BPI Participant.

[R52] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A credential is generated or utilized within the BPI for entities, issuers, or other purposes.
- There is a requirement that the credential holder must prove control over the credential every time it is presented to the BPI or a BPI Participant.

Test Steps:

1. Test Action: Generate or use a credential within the BPI.

Non-Standards Track Work Product

2. Perform an action or transaction within the BPI involving the presentation of the credential.
3. Perform additional actions or transactions within the BPI involving the presentation of the credential.

Expected Results:

1. The BPI should successfully generate or utilize the credential.
2. The credential holder should be able to prove control over the credential as required.
3. The credential holder should be able to consistently prove control over the credential for each action or transaction.

[R53]

Every time a credential utilized in a BPI is used in the BPI by its holder, the BPI MUST verify credential integrity, schema conformance, and that the credential holder is in control of said credential.

Note that credential content verification can only be done through the inspection of underlying documentation or verification by the issuer such as an OpenId Connect Identity Provider [OIDC].

This document will discuss further, more detailed management requirements in the context of BPI participant account management in section 5 [Middleware, Communication and Interoperability](#).

[R53] Testability:

Preconditions:

- Baseline Protocol Implementation (BPI) is installed and configured for testing.
- A credential is generated or utilized within the BPI for entities, issuers, or other purposes.
- There is a requirement that every time the credential is used by its holder in the BPI, the BPI must verify credential integrity, schema conformance, and that the credential holder is in control of the credential.

Test Steps:

1. Generate or use a credential within the BPI.
2. Perform an action or transaction within the BPI involving the presentation of the credential.
3. Perform additional actions or transactions within the BPI involving the presentation of the credential.

Expected Results:

1. The BPI should successfully generate or utilize the credential.
2. The BPI should successfully verify the integrity of the credential, verify schema conformance of the credential, and confirm that the credential holder is in control of the credential.
3. The BPI should consistently verify credential integrity, schema conformance, and control for each action or transaction.

4 BPI Abstraction Layers

BPI Abstraction Layers are the critical umbilical cords of a BPI to its underlying CCSM and external applications such as System of Records or other BPIs.

It is, therefore, critical to carefully craft the requirements on these layers of a BPI in such a way that allows implementers sufficient flexibility, while leveraging established standards and ensuring interoperability through a well-defined, minimal set of interfaces and capabilities.

Since a BPI has two abstraction layers – the BPI and the CCSM Abstraction Layer – the document will define a set of common requirements and differentiate between the two where necessary.

4.1 BPI Abstraction Scope and Components

The Abstraction layers define common standards and processes such as Information Models, APIs and API formats, Process Flows, Roles, Responsibilities, Events, etc. for exposing and managing all BPI capabilities that represent individual steps of the BPI lifecycle processes. This includes but is not limited to:

1. Specifying BPI functional capabilities aligned with already existing common API definitions
2. Onboarding, publishing, upgrading, and retiring BPI APIs and BPI capabilities
3. Coexistence and interoperability with legacy platforms and different BPI stacks

This document defines an Abstraction Layer within the context of a BPI as a set of functions and procedures allowing the interaction of BPI-enabled applications that access the features or data of an operating system, application, or other services with BPI capabilities.

[R54]

BPI Abstraction Layers MUST support Operational Monitoring of an API system.

In the context of this document, an operational monitoring system of BPI APIs refers to the practice of monitoring APIs, most commonly in production, to gain visibility into performance, availability, and functional correctness. These types of systems are designed to help a BPI operator analyze the performance of BPI applications and improve performance. Examples are measurements of how long a service takes to execute, how often it is called, where it is called from, and how much of the total time is spent executing the service.

[R54] Testability:

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes a set of abstraction layers, including Application Layer, Middleware and Communication Layer, Smart Contracts Layer, Data and Privacy Layer, Verifiable Credentials Layer, Consensus and Blockchain Layer, and Network and Transport Layer.
- The BPI includes an API system for communication and interaction with participants.

Test Steps:

1. Examine each of the BPI's abstraction layers to confirm whether they have implemented support for operational monitoring of the API system.
2. Check how the abstraction layers integrate with the operational monitoring features of the API system.
3. Perform interactions or transactions using the BPI components and record the data generated by the operational monitoring features.

Expected Results:

1. Each abstraction layer should include operational monitoring features that allow tracking and analysis of API system performance and behavior.
2. Abstraction layers should seamlessly integrate with the API system's operational monitoring features to provide real-time data and insights.
3. The monitoring system should accurately capture and display operational data, including response times, error rates, and other relevant metrics.

[R55]

BPI Abstraction Layers MUST support an API Portal for provisioning.

A BPI API portal in the context of this document is defined as a visual or a programmatic presentation that provides information about an API at every stage of its lifecycle. A BPI API portal allows operators to expose, document, provision access, and otherwise enable their APIs, and users of those APIs to register applications, reset credentials, provide API feedback, report bugs, etc. A non-normative example of a minimal set of functionalities can be found here [\[API Portal Functionality\]](#)

[R55] Testability:

Non-Standards Track Work Product

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes multiple abstraction layers.
- An API Portal, as described in the requirement, is integrated into the BPI.

Test Steps:

1. Confirm that the BPI's abstraction layers support the integration of the API Portal as described in the requirement.
2. Interact with the API Portal to perform actions such as API provisioning, documentation access, registration of applications, credential resets, providing feedback, and reporting bugs.
3. Review each abstraction layer to confirm that it provides the necessary support and data exchange mechanisms for the API Portal's functionalities

Expected Results:

1. The API Portal is successfully integrated and supported by the abstraction layers.
2. The API Portal functionalities should work as intended, allowing users to perform these actions.
3. Each abstraction layer should support the API Portal's functionalities and data exchange.

[R56]

BPI Abstraction Layers MUST support an API Gateway that does not have Material Impact on BPI latency.

In the context of this document, an API gateway is an application or software pattern that sits in front of an API or a collection of microservices, facilitating requests and delivery of data and services. Its main function is to act as a single entry point and standardized process for interactions between a BPI and its data and services and external/internal users. An API gateway may perform various other functions to support and manage API usage, from authentication to rate limiting to analytics.

In the context of this document, Material Impact refers to something that causes the underlying business requirements of the BPI not to be met. For example in some deployment situations, a 5-second delay can cause transactions to fail or introduce instability to the system, while in other circumstances a 5-minute delay in processing makes no difference to the system as a whole.

[R56] Testability:

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes multiple abstraction layers.
- An API Gateway, as described in the requirement, is integrated into the BPI.

Test Steps:

1. Confirm that the BPI's abstraction layers support the integration of the API Gateway as described in the requirement.
2. Interact with the BPI's APIs and services through the API Gateway, simulating typical usage scenarios, and measure the latency introduced by the API Gateway.
3. Review each abstraction layer to confirm that it provides the necessary support and data exchange mechanisms for the API Gateway without introducing significant latency.

Expected Results:

1. The API Gateway is successfully integrated and supported by the abstraction layers.
2. The measured latency introduced by the API Gateway should be within acceptable limits, ensuring that it does not have a material impact on BPI latency, as defined in the requirement.
3. Each abstraction layer should support the API Gateway's functionalities without causing excessive latency.

[D17]

BPI Abstraction Layers SHOULD support Virtualized APIs.

In the context of this document, virtualized APIs are defined as a production sandbox for continuous integration testing and continuous deployment of APIs.

[D17] Testability:

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes multiple abstraction layers.
- Virtualized APIs, as described in the requirement, are part of the BPI's architecture.

Test Steps:

1. Confirm that the BPI's abstraction layers support the integration of virtualized APIs as described in the requirement.
2. Test the virtualized APIs to verify their functionality for continuous integration testing and continuous deployment.
3. Review each abstraction layer to confirm that they interact effectively with the virtualized APIs, enabling seamless testing and deployment.

Expected Results:

1. The virtualized APIs are successfully integrated and supported by the abstraction layers.
2. The virtualized APIs should provide a production sandbox for continuous integration testing and deployment without introducing issues or errors.
3. Each abstraction layer should support virtualized APIs without causing disruptions to the testing and deployment process.

[D18]

A BPI Abstraction Layer SHOULD support a content delivery network (CDN) (not applicable for a CCSM Abstraction Layer).

In the context of a BPI, a content delivery network is a geographically distributed proxy server network providing high availability and delivery performance of content such as large data files or video streams.

[D18] Testability:

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes multiple abstraction layers.
- CDN support is relevant for the specific abstraction layers being tested.

Test Steps:

1. Confirm that the applicable BPI abstraction layers support the integration of a content delivery network (CDN) as described in the requirement.
2. Perform actions such as requesting large data files or video streams through the CDN to assess the delivery performance and high availability.
3. Review each relevant abstraction layer to confirm that it interacts effectively with the CDN, supporting high availability and efficient content delivery.

Expected Results:

Non-Standards Track Work Product

1. CDN support is successfully integrated and supported by the applicable abstraction layers.
2. The CDN should provide high availability and deliver content with good performance, ensuring efficient content delivery.
3. Each relevant abstraction layer should support CDN functionalities without causing disruptions to content delivery.

[CR13]>[D18]

A CDN utilized in a BPI Abstraction Layer and operated by a 3rd party MUST support BPI subject-specific and time-based content access control.

[CR13]>[D18] Testability:

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes an abstraction layer with integrated CDN support operated by a 3rd party.
- The CDN is configured to support BPI subject-specific and time-based content access control.

Test Steps:

1. Confirm that the CDN, operated by a 3rd party, is integrated into the BPI abstraction layer and configured to support BPI subject-specific and time-based content access control as required.
2. Attempt to access content through the CDN with different subjects and permissions, ensuring that only authorized subjects can access specific content.
3. Attempt to access content through the CDN at different times, ensuring that content availability is restricted based on specified time periods.

Expected Results:

1. The CDN is successfully integrated and configured as specified.
2. The CDN should enforce subject-specific access control, allowing or denying access based on subjects and their permissions.
3. The CDN should enforce time-based access control, allowing or denying access to content depending on the specified time restrictions.

[CR14]>[D18]

A CDN utilized in a BPI Abstraction Layer and operated by a 3rd party MUST support time-based, automated content removal.

[CR14]>[D18] Testability:

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes an abstraction layer with integrated CDN support operated by a 3rd party.
- The CDN is configured to support time-based, automated content removal as required.

Test Steps:

1. Review the documentation and configurations to ensure that the CDN is set up to automatically remove content based on specified time constraints.
2. Upload or access content through the CDN and set time constraints for automatic removal.
3. After the specified time has elapsed, attempt to access the content that was subject to automated removal.

Expected Results:

1. The CDN is successfully integrated and configured as specified.
2. The CDN should automatically remove content based on the specified time constraints, ensuring that content is no longer accessible after the defined time period.
3. The content should no longer be available and should be inaccessible, confirming that automated content removal was successful.

[R57]

BPI Abstraction Layers MUST support integration with internal, as defined in section [5 Middleware, Communication and Interoperability](#), and/or external BPI Subject identity access management (IAM) or identity provider (IdP) systems.

See *Figure 4 as to the meaning of an IdP in a BPI context*, and *Figure 5 in this document as to the meaning of external IAM and its interplay with BPI IAM discussed in section [5 Middleware, Communication and Interoperability](#)*.

[R57] Testability:

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes an abstraction layer.
- Internal or external BPI Subject Identity Access Management (IAM) systems are available and operational, as defined in section 5 Middleware, Communication, and Interoperability.

Test Steps:

1. Confirm that the BPI abstraction layer is capable of integrating with internal and external BPI Subject IAM systems.
2. Perform actions within the BPI that require identity access management, such as user authentication and authorization.
3. Perform actions that require authentication and authorization using external IAM systems.

Expected Results:

1. The BPI abstraction layer is capable of integrating with internal and external BPI Subject IAM systems.
2. The BPI abstraction layer should successfully integrate with internal IAM systems and provide the expected access control.
3. The BPI abstraction layer should successfully integrate with external IAM systems and provide the expected access control.

[R58]

BPI Abstraction Layers MUST support API delivery utilizing the service orchestration capabilities of the BPI Middleware Layer defined in section [5 Middleware, Communication and Interoperability](#).

[R58] Testability

Preconditions:

- A Baseline Protocol Implementation (BPI) is installed and operational.
- The BPI includes an abstraction layer.
- The BPI Middleware Layer, as defined in section 5 Middleware, Communication, and Interoperability, is available and operational.

Test Steps:

1. Confirm that the BPI abstraction layer is integrated with the BPI Middleware Layer, as defined in section 5 Middleware, Communication, and Interoperability.
2. Request and utilize various APIs through the BPI abstraction layer, including actions that require service orchestration.

Expected Results:

Non-Standards Track Work Product

1. The BPI abstraction layer is successfully integrated with the BPI Middleware Layer and can utilize its service orchestration capabilities.
2. The BPI abstraction layer should effectively deliver APIs and utilize service orchestration capabilities, ensuring the proper execution of actions that depend on orchestration.

[R59]

BPI Abstraction Layers MUST support facilitating the discovery and negotiation of capabilities and subsequent integration between a BPI and Legacy Systems/other BPIs as defined in section 5 [Middleware, Communication and Interoperability](#) (for BPI Abstraction Layer only).

[R59] Testability:

Preconditions:

- The BPI Abstraction Layer is operational and correctly integrated into the BPI environment.
- The BPI Abstraction Layer is configured to facilitate the discovery and negotiation of capabilities.
- The BPI Middleware Layer is in place and functioning as defined in section 5 Middleware, Communication, and Interoperability.

Test Steps:

1. Execute a request to the BPI Abstraction Layer to discover available capabilities.
2. Send a request to the BPI Abstraction Layer to negotiate the integration of a specific capability from the list obtained in Step 1.
3. Simulate the integration of a capability with a legacy system.
4. Simulate the integration of a capability with another BPI.

Expected Results:

1. The list from step 1 includes both BPI-specific capabilities and those related to legacy systems and other BPIs.
2. The BPI Abstraction Layer initiates a negotiation process, including specifying requirements, parameters, and any dependencies.
3. The BPI Abstraction Layer successfully mediates the integration by translating requests and responses to and from the legacy system.
4. The BPI Abstraction Layer successfully mediates the integration by handling BPI-specific communication protocols and data formats.

[R60]

BPI Abstraction Layers MUST support the integration of CCSM specific transaction interfaces, transaction crafting, and CCSM specific smart contract management.

Smart Contract management comprises full lifecycle management from testing, initial deployment, updates, and deactivation (for CCSM Abstraction Layer only).

[R60] Testability:

Preconditions:

- The BPI Abstraction Layer is correctly installed and configured for integration with CCSM.
- The CCSM (Core Component Service Manager) is operational and ready for interaction.
- Smart contracts are prepared and accessible for integration.

Test Steps:

1. Send a request to the BPI Abstraction Layer to integrate with CCSM-specific transaction interfaces.
2. Initiate a request for transaction crafting using the BPI Abstraction Layer.
3. Trigger the deployment of a smart contract using the BPI Abstraction Layer.

Expected Results:

1. Transaction interfaces are accessible and functioning correctly.
2. The BPI Abstraction Layer creates transactions with the required parameters and data in a format compatible with CCSM.
3. The BPI Abstraction Layer manages the full lifecycle of the smart contract, including testing, initial deployment, updates, and deactivation. Smart contract updates are performed seamlessly, and deactivated contracts no longer impact the system.

Figure 6 below shows the reference architecture for a BPI or CCSM Abstraction Layer.

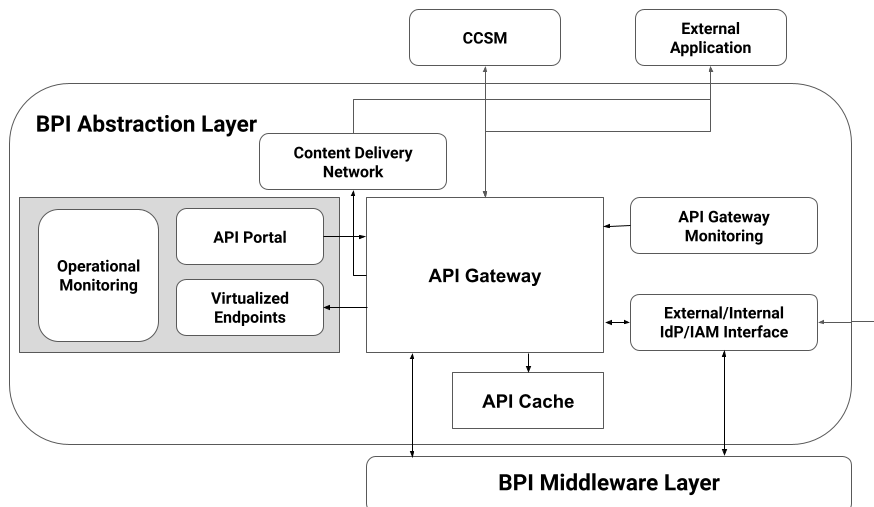


Figure 6: High-Level Reference Architecture of a BPI Layer.

4.2 BPI Abstraction Layer Security and Integration

The security requirements of this section are distinct from the security requirements of the other BPI layers or any custom APIs because the external systems which are invoking services exposed by the BPI or CCSM Abstraction Layer should not be assumed to be a trusted service without authentication. This is because this standard does not define the operating model of external systems or a BPI or any of the BPI layers, and, therefore, must necessarily prescribe requirements assuming a 100% adversarial environment.

[R61]

Non-Standards Track Work Product

Abstraction Layers utilized in a BPI MUST be compatible with widely used external authentication services.

Non-normative examples of such authentication technologies are OAUTH [\[OAuth-2.0\]](#), SAML [\[SAML\]](#), OIDC [\[OIDC\]](#), AD/LDAP [\[ActiveDirectory\]](#).

[\[R61\]](#) Testability:

Preconditions:

- The BPI is set up and configured with the specific Abstraction Layer that is being tested.
- The external authentication services (e.g., OAuth, SAML, OIDC, AD/LDAP) are in place and operational.

Test Steps:

1. Access the BPI system with the specific Abstraction Layer being tested and attempt to configure the Abstraction Layer to use an external authentication service, e.g., OAuth, SAML, OIDC, or AD/LDAP.
2. Configure the Abstraction Layer to use OAuth authentication and attempt to authenticate using valid OAuth credentials.
3. Configure the Abstraction Layer to use SAML authentication and attempt to authenticate using valid SAML credentials.
4. Configure the Abstraction Layer to use OIDC authentication and attempt to authenticate using valid OIDC credentials.
5. Configure the Abstraction Layer to use Active Directory (AD) or LDAP authentication and attempt to authenticate using valid AD/LDAP credentials.

Expected Results:

1. The Abstraction Layer can be configured to use an external authentication service.
2. For all external authentication services, the BPI system successfully authenticates the user.

[\[R62\]](#)

Abstraction Layers utilized in a BPI MUST support roles & access management.

Roles management means that a BPI has the ability to define different roles and associated attributes for BPI Subjects, for example, the role of a "Workgroup Administrator" together with associated attributes such as "Add Workgroup Participant" or "Remove Workgroup Participant".

This document utilizes the NIST definition of Access Management [\[NIST SP 1800-21B\]](#): The set of practices that enables only those permitted the ability to perform an action on a particular resource.

The most common access management approaches are:

- Role Based Access Management (RBAC) tying access rights to a system defined roles and its attributes
- Access Control List (ACL) tying access rights to a table listing the permissions attached to computing resources
- Attribute Based Access Control (ABAC) tying access rights to an evaluation of a set of rules and policies to manage access rights according to specific attributes, such as environmental, system, object, or user information

[\[R62\]](#) Testability:

Preconditions:

- The BPI is set up and configured with the specific Abstraction Layer that is being tested.

Test Steps:

1. Access the BPI system with the specific Abstraction Layer being tested. Create a new role, e.g., "Workgroup Administrator." Define associated attributes for this role, such as "Add Workgroup Participant" and "Remove Workgroup Participant."
2. Assign the newly created "Workgroup Administrator" role to a user or BPI Subject.
3. Create an ACL that lists permissions attached to computing resources within the BPI system. Assign specific access rights to users or roles within the ACL.
4. Define a set of rules and policies based on specific attributes, e.g., environmental, system, object, or BPI Subject information. Assign access rights to users or roles based on the evaluation of these attributes.

Expected Results:

1. The Abstraction Layer provides the ability to create and manage roles and associated attributes.
2. Role-Based Access Management (RBAC) allows users or roles to inherit access rights based on assigned roles and their associated attributes.
3. Users or roles granted access through the ACL can perform the specified actions on computing resources.
4. Users or roles granted access through ABAC can perform actions based on the defined attributes.

[\[R63\]](#)

Abstraction Layers utilized in a BPI MUST support security policy management.

Security policy is defined as a statement of required protection for (a set of) information objects [\[NIST SP 800-192\]](#). An example of a security policy is that only workgroup participants can initiate a workstep within a workflow that is associated with the workgroup.

[\[R63\]](#) Testability:

Preconditions:

- The BPI is set up and configured with the specific Abstraction Layer that is being tested.

Test Steps:

1. Access the BPI system with the specific Abstraction Layer being tested. Create a new security policy, e.g., a policy that only allows workgroup participants to initiate worksteps within a specific workflow associated with a workgroup.
2. Assign the newly created security policy to a specific workflow or a set of information objects within the BPI.
3. Perform an action, such as attempting to initiate a workstep within the workflow associated with a workgroup.
4. Modify the security policy to include different criteria or access rules.
5. Repeat step 3 to test the new security policy.

Expected Results:

1. The Abstraction Layer provides the ability to create and manage security policies.
2. The security policy is successfully applied to the selected workflow or information objects.
3. The BPI enforces the security policy, preventing unauthorized actions.
4. The changes to the security policy are successfully applied.
5. The BPI properly enforces the new security policy and prevents unauthorized actions.

[\[R64\]](#)

Abstraction Layers utilized in a BPI MUST support Single-Sign-On (SSO).

Non-Standards Track Work Product

See [\[SSO\]](#) also for the recommendations of the National Institute of Standards and Technology (NIST Guide to Secure Web Services).

[\[R64\]](#) Testability:

Preconditions:

- The BPI is set up and configured with the specific Abstraction Layer that is being tested.
- SSO provider and identity information have been configured for the BPI.

Test Steps:

1. Access the BPI system with the specific Abstraction Layer being tested.
2. Attempt to access the BPI system and log in using SSO credentials.
3. Ensure that the SSO user's identity is correctly mapped to their corresponding BPI identity or role.
4. Log out of the BPI system.

Expected Results:

1. The Abstraction Layer provides configuration options for SSO settings.
2. SSO is successfully integrated, allowing users to log in using SSO credentials.
3. User mapping between SSO and BPI identities is accurate.
4. Logging out of the BPI also logs the user out of the SSO provider.

[\[R65\]](#)

Abstraction Layers utilized in a BPI MUST support multi-factor authentication.

See the link here for the NIST definition adopted in this document [MFA](#).

[\[R65\]](#) Testability:

Preconditions:

- The BPI is set up and configured with the specific Abstraction Layer that is being tested.
- MFA methods and settings are available and configured within the BPI system.

Test Steps:

1. Access the BPI system with the specific Abstraction Layer being tested.
2. Attempt to enroll a user in MFA.
3. Log in to the BPI system with MFA-enabled credentials.
4. Test the process for MFA recovery and reset.

Expected Results:

1. The BPI allows configuration of MFA settings, and the Abstraction Layer supports multiple MFA methods, such as SMS, email, authenticator apps, and hardware tokens.
2. The Abstraction Layer allows users to set up and configure multiple MFA methods. Users can enroll in MFA with at least two distinct factors.
3. The Abstraction Layer enforces MFA during the login process. Users must successfully complete MFA to access the BPI.
4. Users can regain access if they lose access to one of their MFA factors. Users can reset MFA settings securely.

[\[R66\]](#)

Abstraction Layers utilized in a BPI MUST support hardware security modules (HSM)

This document adopts the [NIST definition](#) and for further information, refer to [HSM](#).

[\[R66\]](#) Testability:

Preconditions:

- The BPI is set up and configured with the specific Abstraction Layer that is being tested.
- HSM devices are available and properly integrated with the Abstraction Layer.

Test Steps:

1. Access the Abstraction Layers configuration settings. Confirm that there are settings or configurations related to external security components and documentation indicating the Abstraction Layers' capability to interact with external security modules.
2. Review the configuration settings or documentation for Abstraction Layers to check for specific configuration parameters related to HSM, such as HSM connection details or API endpoints.
3. Execute basic operations through Abstraction Layers that do not involve HSM.
4. Initiate a test cryptographic operation that requires HSM interaction through Abstraction Layers.
5. Perform cryptographic operations through Abstraction Layers that explicitly require HSM support, such as key generation or signing.
6. Intentionally introduce errors in the HSM configuration or simulate HSM failures.
7. Inspect the logging or auditing mechanisms within Abstraction Layers. Execute HSM-dependent operations and check if corresponding logs are generated. Confirm that logs contain relevant information about the interactions with HSM.

Expected Results:

1. Configuration settings indicate compatibility with external security components. Specific configurations related to HSM are present and documented.
2. The Basic Abstraction Layer has options for HSM integration.
3. Abstraction Layers respond appropriately to standard operations without HSM involvement.
4. Abstraction Layers can communicate with the HSM and receive a response.
5. Cryptographic operations are successful and receive proper HSM support.
6. Abstraction Layers handle errors gracefully, providing meaningful error messages or logging.
7. Logging or auditing mechanisms are functional, and logs capture interactions with HSM.

5 Middleware, Communication, and Interoperability

This section of the document focuses on the concepts and requirements that describe the key capabilities to connect the BPI Abstraction Layer to the BPI Processing Layer and the correctness preserving integration of different BPIs. This section has the following structure:

- BPI Subjects – describing the key capabilities required for access authentication to a BPI, and access authorization of BPI functionality, and BPI- or -user-specific data within a BPI.
- BPI Service Orchestration – describing the middleware capability enabling the invocation of key BPI capabilities through the BPI Abstraction Layer.
- BPI Communication – describing the key capability of how BPI subjects can communicate with one another within a workgroup, within a BPI, and, also, across BPIs.
- BPI Integration – describing the necessary capabilities to ensure the correct intersection of multiple workflows between different BPIs with correctly advanced state across intersecting workflows with minimal or no trust assumptions.

Non-Standards Track Work Product

5.1 BPI Subject Capabilities

A BPI Subject is defined through the capability requirements of section [3 Identifiers, Identity and Credential Management](#). Note that there are two types of BPI Subjects:

- An external BPI Subject that only consumes BPI capabilities
- An internal BPI Subject that manages the provisioning of BPI capabilities to external BPI Subjects, and the integration with other BPIs

This section describes the capabilities of a BPI Subject in the context of a BPI. Unless otherwise differentiated, this document refers to both external and internal BPI Subjects as a BPI Subject.

The minimal set of BPI Subject capabilities are as follows:

[R67]

A BPI Subject MUST be capable of performing all of the following cryptographic key management functionalities:

- Creation
- Derivation
- Storage
- Revocation / rotation
- Backup / recovery

R67 Testability:

Preconditions:

- A BPI system with the necessary cryptographic key management functionality.
- A set of specific cryptographic algorithms and protocols to be used for key management.

Test Steps:

1. A BPI Subject creates a new cryptographic key.
2. Verify that the created key conforms to the specified cryptographic algorithm.
3. A BPI Subject derives a new cryptographic key from an existing key.
4. Ensure that the derived key is different from the original key and conforms to the specified cryptographic algorithm.
5. A BPI Subject stores a cryptographic key in the storage capability of the BPI.
6. Retrieve the stored key and validate its integrity and correctness.
7. A BPI Subject performs key revocation or rotation.
8. Verify that the revoked or rotated key is no longer valid for cryptographic operations.
9. A BPI Subject performs a backup of the cryptographic key.
10. Attempt to recover the key from the backup and ensure its successful restoration.

Passing Criteria:

- The BPI Subject is capable of creating new cryptographic keys.
- The created keys are unique and adhere to the specified cryptographic algorithm.
- The BPI Subject can derive new cryptographic keys from existing keys.
- The derived keys are distinct from the original keys and conform to the specified cryptographic algorithm.
- The BPI Subject is capable of securely storing cryptographic keys.
- The stored keys can be retrieved accurately and maintain their integrity.
- The BPI Subject can successfully perform key revocation or rotation, rendering the revoked or rotated keys invalid.
- The BPI Subject successfully performs a key backup.
- The backup keys can be successfully recovered and restored.

[R68]

A BPI Subject MUST be capable of performing encryption and decryption operations based on BPI specified cryptographic algorithms.

R68 Testability:

Preconditions:

- Set up a BPI test system with the necessary cryptographic algorithms and protocols.
- Ensure that the BPI Subject has access to the encryption and decryption keys.

Test Steps:

1. BPI Subject encrypts a sample plaintext using a supported cryptographic library.
2. Verify that the ciphertext produced by the encryption operation is of the expected length and format.
3. BPI Subject decrypts the ciphertext using the cryptographic library used for encryption.
4. Confirm that the decrypted plaintext matches the original input.
5. BPI Subject encrypts a different plaintext using the same cryptographic library but with a different key.
6. Ensure that the resulting ciphertext is distinct from the previous encryption operation.
7. BPI Subject decrypts the new ciphertext using the corresponding decryption key.
8. Validate that the decrypted plaintext accurately represents the original input.
9. Perform encryption and decryption operations using different cryptographic algorithms supported by the BPI.
10. Confirm that the encrypted and decrypted data remains consistent and conforms to the specified algorithms.

Passing Criteria:

- The BPI Subject successfully encrypts plaintext into ciphertext.
- The ciphertext generated by the encryption operation meets the expected format and length.
- The BPI Subject is capable of decrypting ciphertext back into the original plaintext.
- The decrypted plaintext matches the original input, indicating accurate decryption.
- Encryption of different plaintexts with different keys produces distinct ciphertexts.
- Decryption of the distinct ciphertexts results in their respective original plaintexts.
- The encrypted and decrypted data remains consistent across different cryptographic algorithms.

[R69]

A BPI Subject MUST be capable of performing cryptographic hashing operations based on BPI specified cryptographic algorithms.

R69 Testability:

Preconditions:

- Set up a BPI system with cryptographic hashing algorithms.
- Ensure that the BPI Subject has access to the cryptographic hashing functions.

Non-Standards Track Work Product

Test Steps:

1. Generate an input data sample.
2. BPI Subject applies the cryptographic hashing function to input data from the sample.
3. Verify that the hash output has the expected length and format.
4. BPI Subject changes a single character in the input data and repeats the hashing operation.
5. Confirm that the resulting hash value is different from the previous hash.
6. BPI Subject performs multiple hashing operations using different input data from the sample.
7. Validate that each input produces a unique hash output.
8. Utilize different cryptographic hashing algorithms supported by the BPI.
9. BPI Subject repeats steps 2. through 7.

Passing Criteria:

- The BPI Subject successfully performs the cryptographic hashing operation.
- The generated hash output has the expected length and format.
- Changing a single character in the input data results in a different hash value.
- Each input data from the generated sample produces a unique hash output.
- The BPI supports multiple cryptographic hashing algorithms that produce algorithm compliant outputs (expected length and format, hash changes when input changes)

[R70]

A BPI Subject MUST be capable of performing cryptographic digital signature creation and verification based on BPI specified cryptographic algorithms.

R70 Testability:

Preconditions:

- Set up a BPI system with cryptographic algorithms and protocols.
- Ensure that the BPI Subject has access to the necessary cryptographic functions for digital signature creation and verification.

Test Steps:

1. BPI Subject generates a key pair for digital signature.
2. BPI Subject creates a sample message or data to be signed.
3. BPI Subject digitally signs the message with the private key.
4. BPI Subject verifies the digital signature using the corresponding public key and the signed message.
5. BPI Subject modifies a single character in the signed message and attempts to verify the digital signature.
6. BPI Subject confirms that the verification fails, indicating the integrity of the message has been compromised.
7. BPI Subject generates multiple key pairs and signs the same message with different private keys.
8. BPI Subject verifies each digital signature using the corresponding public key and confirms the validity of each signature.
9. BPI Subject repeats steps 1. through 8. for all cryptographic algorithms supported by the BPI for digital signature creation.

Passing Criteria:

- The BPI Subject successfully generates a key pair for digital signature.
- The BPI Subject accurately creates a digital signature for the given message.
- The digital signature can be successfully verified by the BPI Subject using the corresponding public key and the signed message.
- Modifying a single character in the signed message causes the verification to fail.
- Multiple digital signatures from different key pairs are all successfully verified.
- The BPI Subject can create digital signatures using multiple cryptographic algorithms.
- The digital signatures from different cryptographic algorithms can be verified using the appropriate verification algorithm.

[R71]

A BPI Subject MUST be capable of performing cryptographic threshold-digital-signature creation and verification based on BPI specified cryptographic algorithms.

R71 Testability:

Preconditions:

- Set up a BPI system with cryptographic algorithms and protocols.
- Ensure that BPI Subjects have access to the necessary cryptographic functions for threshold digital signature creation and verification.
- Establish a group of participants (BPI Subjects) who will be involved in the threshold digital signature scheme.
- Establish a digital signature threshold condition.

Test Steps:

1. Generate a threshold key pair for digital signature for each participant.
2. Create a sample message or data to be signed.
3. Perform a threshold digital signature creation process with a subset of participants.
4. Combine the partial signatures to generate the complete threshold digital signature.
5. Verify the threshold digital signature using the combined public keys of the participating subjects.
6. Modify a single character in the signed message and attempt to verify the threshold digital signature.
7. Confirm that the verification fails, indicating the integrity of the message has been compromised.
8. Add or remove participants from the group and repeat the threshold digital signature creation and verification process.
9. Repeat steps 1. through 8. for all cryptographic algorithms supported by the BPI for threshold digital signature creation.

Passing Criteria:

1. Successfully generate a threshold key pair for digital signature for each participant.
2. Accurately perform the threshold digital signature creation process using the subset of participants.
3. Combine the partial signatures to generate the complete threshold digital signature correctly.
4. The threshold digital signature can be successfully verified using the combined public keys of the participating subjects.
5. Modifying a single character in the signed message causes the verification to fail.
6. Participants can be added or removed from the group without affecting the threshold digital signature generation and verification process.
7. Threshold digital signatures from different cryptographic algorithms can be verified using the appropriate verification algorithm.

[R72]

A BPI Subject MUST be capable of performing cryptographic secret sharing based on BPI specified cryptographic algorithms between two or more BPI Subjects.

An example of cryptographic secret sharing is called "Shamir Secret Sharing" see [\[Shamir\]](#).

R72 Testability:

Preconditions:

Non-Standards Track Work Product

- Set up a BPI system with cryptographic algorithms and protocols.
- Ensure that the BPI Subjects have access to the necessary cryptographic functions for secret sharing.
- Establish a group of two or more BPI Subjects to perform the secret sharing process.

Test Steps:

1. Generate a secret value to be shared.
2. Select the number of shares and the threshold value for the secret sharing scheme.
3. Perform the secret sharing process by dividing the secret value into shares.
4. Distribute the shares to the participating BPI Subjects.
5. Reconstruct the secret value using the required number of shares (threshold).
6. Verify that the reconstructed secret value matches the original secret value.
7. Modify one of the shares and attempt to reconstruct the secret value.
8. Confirm that the reconstruction fails, indicating the integrity of the secret has been compromised.
9. Change the number of shares or the threshold value and repeat the secret sharing and reconstruction process.
10. Repeat steps 1. through 9. for all cryptographic algorithms supported by the BPI for secret sharing.

Passing Criteria:

Independent of the applied secret sharing algorithm:

- The BPI successfully generates a secret value to be shared.
- The BPI accurately performs the secret sharing process by dividing the secret into shares.
- The shares are distributed correctly to the participating BPI Subjects.
- The BPI successfully reconstructs the secret value using the required number of shares.
- The reconstructed secret value matches the original secret value.
- Modifying one of the shares causes the reconstruction process to fail.
- Changing the number of shares or the threshold value does not affect the secret sharing and reconstruction process.

[R73]

An external BPI Subject MUST at least be able to create, read, update and delete the following BPI core components following business rules for each component that were established by the BPI operator and agreed to by BPI Subjects:

- A BPI Account belonging to the BPI Subject. Note that a BPI Account as defined in section [6.4 BPI Account](#) is different from a BPI Subject Account as defined in section [5.2 BPI Subject Account](#)
- A BPI Workgroup as defined in section [6.3 BPI Workgroup](#)
- A BPI Workflow as defined in section [6.2 BPI Workflow](#)
- A BPI Workstep as defined in section [6.1 BPI Workstep](#)
- A BPI Transaction as defined in section [6.5 BPI Transactions](#)

There may be other BPI components such as role definitions or security policies. The enablement of additional components in a BPI beyond this standard is left to each specific implementation.

[R73](#) Testability:

Preconditions:

- Ensure that a BPI system is set up and functional.
- Establish business rules for each BPI core component (Account, Workgroup, Workflow, Workstep, Transaction).
- Confirm that the external BPI Subject has the necessary permissions and access rights to create, read, update, and delete BPI core components.

Test Steps:

1. Create a new BPI Account for the external BPI Subject.
2. Verify that the BPI Account is successfully created and associated with the BPI Subject.
3. Read the details of the created BPI Account and validate that the information is accurate.
4. Update the BPI Account by modifying one or more of its attributes (e.g., name, contact information).
5. Confirm that the changes to the BPI Account are successfully updated and reflected in the system.
6. Create a new BPI Workgroup and associate it with the BPI Account.
7. Verify that the BPI Workgroup is created and linked to the BPI Account.
8. Read the details of the created BPI Workgroup and validate that the information is accurate.
9. Update the BPI Workgroup by modifying its attributes (e.g., name, description).
10. Confirm that the changes to the BPI Workgroup are successfully updated and reflected in the system.
11. Create a new BPI Workflow within the BPI Workgroup.
12. Verify that the BPI Workflow is created and linked to the corresponding BPI Workgroup.
13. Read the details of the created BPI Workflow and validate that the information is accurate.
14. Update the BPI Workflow by modifying its attributes (e.g., name, steps).
15. Confirm that the changes to the BPI Workflow are successfully updated and reflected in the system.
16. Create a new BPI Workstep within the BPI Workflow.
17. Verify that the BPI Workstep is created and associated with the corresponding BPI Workflow.
18. Read the details of the created BPI Workstep and validate that the information is accurate.
19. Update the BPI Workstep by modifying its attributes (e.g., name, instructions).
20. Confirm that the changes to the BPI Workstep are successfully updated and reflected in the system.
21. Create a new BPI Transaction within the BPI Workstep.
22. Verify that the BPI Transaction is created and linked to the corresponding BPI Workstep.
23. Read the details of the created BPI Transaction and validate that the information is accurate.
24. Update the BPI Transaction by modifying its attributes (e.g., data, status).
25. Confirm that the changes to the BPI Transaction are successfully updated and reflected in the system.
26. Delete the BPI Transaction.
27. Verify that the BPI Transaction is no longer present in the system.
28. Delete the BPI Workstep.
29. Verify that the BPI Workstep is successfully removed from the system.
30. Delete the BPI Workflow.
31. Verify that the BPI Workflow is no longer present in the system.
32. Delete the BPI Workgroup.
33. Verify that the BPI Workgroup is successfully deleted.
34. Delete the BPI Account.
35. Verify that the BPI Account is no longer accessible in the system.

Passing Criteria:

- The external BPI Subject can create a new BPI Account and associate it with their identity.
- The external BPI Subject can read the details of the created BPI Account accurately.
- The external BPI Subject can update the BPI Account and confirm the changes are reflected in the system.
- The external BPI Subject can create a new BPI Workgroup and associate it with the BPI Account.

Non-Standards Track Work Product

- The external BPI Subject can read the details of the created BPI Workgroup accurately.
- The external BPI Subject can update the BPI Workgroup and confirm the changes are reflected in the system.
- The external BPI Subject can create a new BPI Workflow within the BPI Workgroup.
- The external BPI Subject can read the details of the created BPI Workflow accurately.
- The external BPI Subject can update the BPI Workflow and confirm the changes are reflected in the system.
- The external BPI Subject can create a new BPI Workstep within the BPI Workflow.
- The external BPI Subject can read the details of the created BPI Workstep accurately.
- The external BPI Subject can update the BPI Workstep and confirm the changes are reflected in the system.
- The external BPI Subject can create a new BPI Transaction within the BPI Workstep.
- The external BPI Subject can read the details of the created BPI Transaction accurately.
- The external BPI Subject can update the BPI Transaction and confirm the changes are reflected in the system.
- The external BPI Subject can delete the BPI Transaction successfully.
- The BPI Transaction is no longer accessible after deletion.
- The external BPI Subject can delete the BPI Workstep successfully.
- The BPI Workstep is no longer accessible after deletion.
- The external BPI Subject can delete the BPI Workflow successfully.
- The BPI Workflow is no longer accessible after deletion.
- The external BPI Subject can delete the BPI Workgroup successfully.
- The BPI Workgroup is no longer accessible after deletion.
- The external BPI Subject can delete the BPI Account successfully.
- The BPI Account is no longer accessible after deletion.

[R74]

An internal BPI Subject MUST be able to create, read, update and delete all components of a BPI.

[R74](#) Testability:

Preconditions:

- Ensure that the BPI system is set up and functional.
- Authenticate and authorize the internal BPI Subject with appropriate permissions and access rights to create, read, update, and delete BPI components.

Test Steps:

1. All test steps from [R73](#) Testability for an internal BPI Subject.
2. For any other BPI component type in the BPI:
 - Create a new BPI component.
 - Verify that the BPI component is created.
 - Read the details of the created BPI component and validate that the information is accurate.
 - Update the BPI component by modifying its attributes (e.g., name, description).
 - Confirm that the changes to the BPI component are successfully updated and reflected in the system.
 - Delete the BPI component.
 - Verify that the BPI Transaction is no longer present in the system.

Passing Criteria:

- All test passing criteria from [R73](#) Testability for an internal BPI subject.
- For any other BPI component type in the BPI:
 - The internal BPI Subject can create a new BPI component.
 - The internal BPI Subject can read the details of the created BPI component accurately.
 - The internal BPI Subject can update the BPI component and confirm the changes are reflected in the system.
 - The BPI component is no longer accessible after deletion by the internal BPI Subject.

[R75]

A BPI Subject MUST be able to send and receive BPI messages from other BPI subjects using the BPI Communication capability as defined in section [5.3 BPI Service Orchestration](#).

[R75](#) Testability:

Preconditions:

- A BPI system is operational and the BPI Communication capability is enabled.
- BPI Subjects are registered and authenticated within the BPI system.

Test Steps:

1. Send a BPI message from the source BPI Subject to a target BPI Subject using the BPI Communication capability.
2. Verify that the BPI message is successfully sent and delivered to the target BPI Subject.
3. Read and validate the content of the received BPI message to ensure its integrity and correctness.
4. Reply to the received BPI message from the target BPI Subject to the source BPI Subject.
5. Confirm that the reply message is successfully sent and received by the source BPI Subject.
6. Read and verify the content of the reply message to ensure its accuracy.

Passing Criteria:

- The BPI Subject can successfully send BPI messages to other BPI Subjects.
- BPI messages are received and delivered accurately to the target BPI Subjects.
- The content of the received BPI messages is correct and matches the sent messages.
- Replies to received BPI messages are sent and received correctly.

[R76]

A BPI Subject MUST be able to create, read, and delete BPI messages from other BPI subjects to said BPI Subject using the BPI Communication capability as defined in section [5.3 BPI Service Orchestration](#).

Note that when a BPI Subject executes these capabilities they are understood to be enabled by services under the direct custody of the BPI Subject. However, the BPI Subject may delegate these capabilities to the BPI in which the BPI subject operates.

[R76](#) Testability:

Preconditions:

- A BPI system is operational and the BPI Communication capability is enabled.
- BPI Subjects are registered and authenticated within the BPI system.

Test Steps:

Non-Standards Track Work Product

1. Create a BPI message from a source BPI Subject to the target BPI Subject using the BPI Communication capability.
2. Verify that the BPI message is successfully created and can be accessed by the target BPI Subject.
3. Read the content of the received BPI message to ensure its accuracy and completeness.
4. Delete the BPI message from the target BPI Subject using the BPI Communication capability.
5. Confirm that the BPI message is successfully deleted and no longer accessible by the target BPI Subject.
6. Create multiple BPI messages from different source BPI Subjects to the same target BPI Subject.
7. Verify that all the created BPI messages can be accessed and read by the target BPI Subject.
8. Delete specific BPI messages from the target BPI Subject.
9. Confirm that the deleted BPI messages are successfully removed and no longer available to the target BPI Subject.
10. Attempt to read deleted BPI messages from the target BPI Subject.
11. Ensure that the deleted BPI messages are no longer accessible and return an appropriate error or empty response.

Passing Criteria:

- The BPI Subject can successfully create BPI messages from other BPI Subjects using the BPI Communication capability.
- Created BPI messages are received and accessible by the target BPI Subject.
- The content of the received BPI messages is accurate and complete.
- BPI messages can be deleted from the target BPI Subject using the BPI Communication capability.
- Deleted BPI messages are successfully removed and no longer accessible by the target BPI Subject.
- Multiple BPI messages from different sources can coexist and be accessed by the target BPI Subject.
- Attempting to read deleted BPI messages results in an appropriate error or empty response.

[R77]

A BPI Subject MUST be able to delegate one or more BPI Subject capabilities to a 3rd party that is also a BPI Subject.

[R77](#) Testability:

Preconditions:

- A BPI system is operational.
- The BPI Subject and the 3rd party BPI Subject are registered and authenticated within the BPI system.

Test Steps:

1. Identify the BPI Subject's capabilities that can be delegated to a 3rd party BPI Subject.
2. Delegate one or more capabilities from the BPI Subject to the 3rd party BPI Subject.
3. Verify that the delegation process is successful and the capabilities are transferred to the 3rd party BPI Subject.
4. Confirm that the BPI Subject no longer has access to the delegated capabilities.
5. Test the delegated capabilities by performing actions associated with those capabilities using the 3rd party BPI Subject's credentials.
6. Ensure that the 3rd party BPI Subject can successfully perform the delegated capabilities.
7. Revoke the delegation of the capabilities from the 3rd party BPI Subject.
8. Verify that the revocation process is successful and the capabilities are no longer available to the 3rd party BPI Subject.
9. Attempt to use the revoked capabilities using the 3rd party BPI Subject's credentials.
10. Confirm that the 3rd party BPI Subject no longer has access to the revoked capabilities.

Passing Criteria:

- The BPI Subject can delegate one or more capabilities to a 3rd party BPI Subject.
- The delegation process is successful, and the capabilities are transferred to the 3rd party BPI Subject.
- The BPI Subject no longer has access to the delegated capabilities.
- The 3rd party BPI Subject can successfully perform the delegated capabilities.
- The revocation of delegated capabilities from the 3rd party BPI Subject is successful.
- The 3rd party BPI Subject no longer has access to the revoked capabilities.

5.2 BPI Subject Account

A BPI Subject Account is a key component of a BPI because it is the anchor point for all important BPI functions – authentication and authorization, messages, workgroups, workflows, worksteps, transactions etc. Without the notion of a cryptographically verifiable state of a BPI Subject within a BPI – afforded by a BPI Subject Account – no other BPI functions will be able to operate successfully, since all functions are predicated on cryptographically verifiable counterparties that maintain a referable and verifiable state within a BPI.

The following requirements on a BPI Subject Account assume that all identifiers and credentials referring to a BPI Subject are compliant with the requirements in section [6 Identifiers, Identity and Credential Management](#).

[R78]

A BPI Subject Account MUST NOT contain any personal identifiable information (PII).

This requirement facilitates compliance with privacy laws in different jurisdictions.

[R78](#) Testability:

Preconditions:

- A BPI system is operational.
- A BPI Subject Account is created and accessible within the BPI system.

Test Steps:

1. Access the BPI Subject Account and inspect the stored information.
2. Verify that no personal identifiable information (PII) is present in the BPI Subject Account.
3. Check all fields, attributes, or properties associated with the BPI Subject Account for any signs of PII.
4. Validate that sensitive information such as full name, date of birth, social security number, address, contact details, or any other personally identifiable information is not stored within the BPI Subject Account.
5. Attempt to update the BPI Subject Account by entering information suspected to be PII.
6. Verify that the system detects the suspected PII and prevents the update from being applied immediately.
7. Confirm that the system places the update with suspected PII in a pending state for further review.
8. Review the pending updates and ensure they are manually inspected to determine if they contain PII.
9. If the updates do not contain PII, approve and apply the changes to the BPI Subject Account.
10. If the updates contain PII, reject the changes and provide appropriate guidance to the user for removing or anonymizing the PII.
11. Repeat steps 5-10 with different types of information that may be considered PII.

Passing Criteria:

- The BPI Subject Account does not contain any personal identifiable information (PII).
- No sensitive information or PII is present in the fields, attributes, or properties of the BPI Subject Account.

Non-Standards Track Work Product

- The system detects and prevents the addition of PII to the BPI Subject Account.
- Updates suspected to contain PII are placed in a pending state for manual review.
- Updates without PII are approved and applied to the BPI Subject Account.

[R79]

A BPI Subject Account MUST NOT be created by an external BPI Subject.

This requirement ensures that only a BPI operator can create a BPI Subject Account to avoid account creation spamming and implementation of KYC processes required in certain jurisdictions.

R79 Testability:

Preconditions:

- A BPI system is operational.
- An external BPI Subject can access the BPI.

Test Steps:

1. Attempt to create a BPI Subject Account using the credentials of an external BPI Subject.
2. Verify that the system prevents the creation of a BPI Subject Account by an external BPI Subject.
3. Check for any error messages or notifications indicating that the creation of a BPI Subject Account by an external BPI Subject is not allowed.
4. Validate that the system does not generate any records or data associated with the attempted creation of a BPI Subject Account by an external BPI Subject.

Passing Criteria:

- The system prohibits the creation of a BPI Subject Account by an external BPI Subject.
- An appropriate error message or notification is displayed indicating that the creation of a BPI Subject Account by an external BPI Subject is not allowed.
- No records or data related to the attempted creation of a BPI Subject Account by an external BPI Subject are generated.

[R80]

A BPI Subject Account MUST have at least the following data properties:

1. A unique, resolvable, and cryptographically verifiable identifier
2. One or more security policies including authentication and authorization policies
3. A cryptographically verifiable credential establishing the (legal) identity of a BPI Subject, and utilized by a BPI in the creation of the BPI Subject Account
4. A BPI Subject Account recovery key that can be only derived by the BPI Subject owning the BPI Subject Account, and is independent of the private key(s) associated with the BPI Subject Account identifier.
5. A list of all BPI Accounts related to the BPI Subject Account

This minimal set of requirements ensures:

- *that access to a BPI Subject account can be cryptographically verified ([R80.1]),*
- *that a BPI account can fine-grain access and authorization requirements for the BPI Subject ([R80.2]),*
- *that other BPI Subjects can independently verify the identity used to establish the BPI Subject Account ([R80.3]),*
- *that a BPI Subject can independently recover a BPI Subject Account even if their private key was compromised, and ensure forward security ([R80.4]), and*
- *that there is a provable relationship with BPI Accounts connected to BPI state objects connected to the BPI Subject Account owner ([R80.5]).*

R80 Testability:

Preconditions:

- BPI system is operational.
- A BPI Subject can access the BPI.

Test Steps:

1. Create a BPI Subject Account by providing the necessary data properties, including a unique, resolvable, and cryptographically verifiable identifier, one or more security policies, a cryptographically verifiable credential, a BPI Subject Account recovery key, and a list of related BPI Accounts.
2. Verify that the BPI Subject Account is successfully created without any errors or notifications.
3. Retrieve the BPI Subject Account and validate that the data properties are correctly stored and associated with the account.
4. Perform authentication and authorization checks using the security policies associated with the BPI Subject Account, and verify that the account behaves as expected.
5. Update the BPI Subject Account recovery key and ensure that the key is successfully changed and can only be derived by the BPI Subject owning the account.
6. Add or remove BPI Accounts from the BPI Subject Account's list of related accounts and confirm that the changes are reflected accurately.
7. Attempt to create a BPI Subject Account with one or more of the required data properties missing.
8. Verify that a BPI Subject Account is not created and an appropriate error message is displayed.

Passing Criteria:

- A BPI Subject Account is created without any errors or notifications.
- The created BPI Subject Account contains all the specified data properties, including a unique identifier, security policies, a cryptographically verifiable credential, a BPI Subject Account recovery key, and a list of related BPI Accounts.
- The security policies associated with the BPI Subject Account are enforced correctly during authentication and authorization processes.
- The BPI Subject Account recovery key can be updated and is only derivable by the BPI Subject owning the account.
- The list of related BPI Accounts in the BPI Subject Account is accurately maintained and can be modified as intended.
- A BPI Subject Account is not created if one or more of the required data properties are missing.

[R81]

A BPI Subject Account MUST have a cryptographically verifiable audit trail for all BPI Subject Account operations both attempted and completed.

This is a critical audit requirement to ensure ease of compliance with regulatory statutes across jurisdictions. Audit trail in the context of this document refers to the time sequenced capture of the state of a data structure within a BPI such as a BPI Subject Account from state instantiation to state deletion or archiving including all successful and unsuccessful state changes. An audit trail is said to be cryptographically verifiable if the audit trail includes auxiliary cryptographic information that allows 3rd parties to verify an audit trail as consistent. Creating a linked chain of hashes of each audit trail entry such as a Merkle proof is an example of such auxiliary cryptographic information.

R81 Testability:

Preconditions:

- A BPI system is operational.
- A BPI Subject Account exists.
- A cryptographic technique to ensure the cryptographic verifiability of the audit trail.

Test Steps:

1. Perform various operations on the BPI Subject Account, such as creating, updating, and deleting account data.

Non-Standards Track Work Product

2. Retrieve the audit trail associated with the BPI Subject Account.
3. Verify that the audit trail contains a record for each operation performed on the BPI Subject Account, including both attempted and completed operations.
4. Validate that the audit trail records include relevant information such as the type of operation, timestamp, BPI Subject involved, and any additional details specific to the operation.
5. Verify the integrity of the audit trail by ensuring that it is cryptographically verifiable, preventing tampering or unauthorized modifications based on the chosen cryptographic technique.
6. Perform a simulated attempt to tamper with the audit trail and confirm that the system detects the tampering and maintains the integrity of the audit trail.
7. Verify that the audit trail accurately reflects the sequence of operations and their outcomes, including successful and failed attempts.
8. Test the search and retrieval functionality of the audit trail by querying specific operations and confirming that the corresponding records are returned accurately.

Passing Criteria:

- The BPI Subject Account supports various operations, including creation, update, and deletion of account data.
- The audit trail associated with the BPI Subject Account contains a record for each attempted and completed operation.
- The audit trail records include relevant information and details about the operations performed on the BPI Subject Account.
- The audit trail is cryptographically verifiable, ensuring the integrity of the recorded operations.
- The system detects any tampering attempts on the audit trail and maintains its integrity.
- The audit trail accurately reflects the sequence of operations and their outcomes.
- The search and retrieval functionality of the audit trail returns the expected records accurately.

[\[R82\]](#)

A BPI Subject Account **MUST** be encrypted to and decrypted by a cryptographic key only known to the BPI Subject Account owner.

[R82](#) Testability:

Preconditions:

- A BPI system is operational.
- A BPI Subject Account exists.

Test Steps:

1. Retrieve the BPI Subject Account from the BPI system.
2. Attempt to access the encrypted data of the BPI Subject Account without the cryptographic key known only to the BPI Subject Account owner.
3. Verify that the encrypted data cannot be decrypted or accessed without the correct cryptographic key.
4. Obtain the cryptographic key known only to the BPI Subject Account owner.
5. Decrypt the encrypted data of the BPI Subject Account using the cryptographic key.
6. Validate that the decrypted data matches the expected data associated with the BPI Subject Account.
7. Attempt to decrypt the encrypted data using an incorrect cryptographic key.
8. Verify that the decryption fails and the data remains inaccessible.
9. Update the encrypted data of the BPI Subject Account with new information.
10. Ensure that the updated data is encrypted using the cryptographic key known only to the BPI Subject Account owner.
11. Retrieve the BPI Subject Account and verify that the encrypted data has been successfully updated.

Passing Criteria:

- The encrypted data of the BPI Subject Account cannot be accessed or decrypted without the correct cryptographic key.
- When the correct cryptographic key is provided, the encrypted data can be successfully decrypted and accessed.
- Decrypting the encrypted data using an incorrect cryptographic key fails to provide access to the data.
- The updated data of the BPI Subject Account is successfully encrypted using the cryptographic key.

[\[D19\]](#)

A BPI Subject Account **SHOULD** be encrypted when not in active use by the BPI Subject that owns the BPI Subject Account.

[D19](#) Testability:

Preconditions:

- A BPI system is operational.
- A BPI Subject Account exists.

Test Steps:

1. Retrieve the BPI Subject Account from the BPI system.
2. Verify that the BPI Subject Account is correctly decrypted when it is accessed.
3. Attempt to access the encrypted data of the BPI Subject Account when it is not in active use.
4. Verify that the BPI Subject Account data is encrypted.

Passing Criteria:

- The BPI Subject Account is encrypted when not in active use by the BPI Subject.
- The decrypted data matches the expected data associated with the BPI Subject Account.

[\[R82\]](#) and [\[D19\]](#) ensure that a compromised BPI does not expose BPI Subject Account information. A BPI may use the BPI Subject Account unique identifier or another unique identifier such as an account number known to the BPI Subject Account holder as an unencrypted identifier of the BPI Subject Account stored within the BPI.

[\[R83\]](#)

A BPI Subject Account owner **MUST** be able to perform the following operations on its BPI Subject Account: read, update and delete.

[R83](#) Testability:

Preconditions:

- A BPI system is operational.
- A BPI Subject Account exists.
- A BPI Subject Account is owned by the account owner.

Test Steps:

1. Authenticate the BPI Subject Account owner.
2. Retrieve and read the BPI Subject Account.
3. Verify that the retrieved BPI Subject Account contains the expected data.
4. Update the BPI Subject Account by modifying its properties.
5. Retrieve and read the updated BPI Subject Account.
6. Verify that the retrieved BPI Subject Account reflects the applied updates.
7. Delete the BPI Subject Account.

Non-Standards Track Work Product

8. Attempt to retrieve the deleted BPI Subject Account.
9. Verify that the deleted BPI Subject Account cannot be accessed.

Passing Criteria:

- The BPI Subject Account owner can read and retrieve their BPI Subject Account.
- The retrieved BPI Subject Account contains the expected data.
- The BPI Subject Account owner can update their BPI Subject Account, and the updates are reflected in the retrieved account.
- The BPI Subject Account owner can delete their BPI Subject Account.
- The deleted BPI Subject Account is inaccessible and cannot be retrieved.

[R84]

An internal BPI Subject MUST be able to perform the following operations on its BPI Subject Account: create, read, update and delete.

[R84](#) Testability:

Preconditions:

- A BPI system is operational.
- An internal BPI Subject exists.

Test Steps:

1. Authenticate the internal BPI Subject.
2. Create a new BPI Subject Account for the internal BPI Subject.
3. Retrieve and read the BPI Subject Account.
4. Verify that the retrieved BPI Subject Account contains the expected data.
5. Update the BPI Subject Account by modifying its properties.
6. Retrieve and read the updated BPI Subject Account.
7. Verify that the retrieved BPI Subject Account reflects the applied updates.
8. Delete the BPI Subject Account.
9. Attempt to retrieve the deleted BPI Subject Account.
10. Verify that the deleted BPI Subject Account cannot be accessed.

Passing Criteria:

1. The internal BPI Subject can create a new BPI Subject Account.
2. The internal BPI Subject can read and retrieve its BPI Subject Account.
3. The retrieved BPI Subject Account contains the expected data.
4. The internal BPI Subject can update its BPI Subject Account, and the updates are reflected in the retrieved account.
5. The internal BPI Subject can delete its BPI Subject Account.
6. The deleted BPI Subject Account is inaccessible and cannot be retrieved.

[R85]

An internal BPI Subject MUST be able to perform the following operations on other BPI Subject Accounts: create, read, update and delete provided that there exist one or more security policies authorizing these capabilities to said internal BPI Subject.

[R85](#) Testability:

Preconditions:

- A BPI system is operational.
- There are BPI Subject Accounts existing.
- The internal BPI Subject has the necessary authorization based on security policies to perform BPI Subject Account operations.

Test Steps:

1. Authenticate the internal BPI Subject against one of the BPI Subject Accounts for which the internal BPI Subject is authorized based on an existing security policy in said BPI Subject Account.
2. Retrieve and read an existing BPI Subject Account.
3. Verify that the retrieved BPI Subject Account contains the expected data.
4. Update an existing BPI Subject Account by modifying its properties.
5. Retrieve and read the updated BPI Subject Account.
6. Verify that the retrieved BPI Subject Account reflects the applied updates.
7. Delete an existing BPI Subject Account.
8. Verify that the BPI Subject Account is successfully deleted.
9. Attempt to retrieve the deleted BPI Subject Account.
10. Verify that the deleted BPI Subject Account cannot be accessed.

Passing Criteria:

- An existing BPI Subject Account exists and can be retrieved by an internal BPI Subject with the correct authorization based on an existing security policy.
- The internal BPI Subject can read and retrieve an existing BPI Subject Account.
- The retrieved BPI Subject Account contains the expected data.
- The internal BPI Subject can update an existing BPI Subject Account, and the updates are reflected in the retrieved account.
- The internal BPI Subject can delete an existing BPI Subject Account successfully.
- The deleted BPI Subject Account is inaccessible and cannot be retrieved.

[R86]

A BPI Subject Account owner MUST be notified by a BPI of any changes to a BPI Subject Account or BPI Accounts associated with a BPI Subject Account.

[R86](#) Testability:

Preconditions:

- A BPI system is operational.
- A BPI Subject Account exists.
- The BPI Subject Account owner is logged in and actively using the BPI system.

Test Steps:

1. Perform a change to the BPI Subject Account, such as updating a property or adding/removing an associated BPI Account.
2. Verify that the BPI Subject Account owner receives a notification regarding the change.
3. Access the notification received by the BPI Subject Account owner.

Non-Standards Track Work Product

4. Verify that the notification contains accurate information about the change made to the BPI Subject Account or the associated BPI Accounts.

Passing Criteria:

- The BPI Subject Account owner is promptly notified of any changes made to the BPI Subject Account or the associated BPI Accounts.
- The notification is successfully delivered to the BPI Subject Account owner.
- The notification received by the BPI Subject Account owner contains accurate and relevant information about the change made.
- The BPI Subject Account owner can access and view the notification without any issues.

5.3 BPI Service Orchestration

This document defines service orchestration as the sequence and conditions organized in an interaction pattern that one or more service agents must follow, and in which one service invokes other services to achieve its desired goal.

BPI service orchestration is intended to operate in a business environment with high service volume (many BPI service requests), low system latency (fast completion of BPI service requests by a BPI required), and where BPI service requests change BPI data often. It is key to achieve a flexible, loosely-coupled architecture as described in section [2.8 Baseline Protocol Reference Architecture](#). BPI service orchestration is geared towards high-volume, low latency environments with many data changes.

[R87]

BPI service orchestration utilized in a BPI MUST be able to identify the actions to be completed by services based on message context and content, and successfully orchestrate the desired action.

[R87](#) Testability:

Preconditions:

- A BPI system is operational.
- BPI services and their corresponding actions are properly configured and available.

Test Steps:

1. Send a message with a specific context and content to the BPI service orchestration.
2. Verify that the BPI service orchestration correctly identifies the actions to be completed based on the message context and content.
3. Confirm that the identified actions align with the desired actions for the given message.
4. Execute the identified actions through the BPI service orchestration.
5. Validate that the desired actions are successfully orchestrated by the BPI service orchestration.
6. Verify that any required services are invoked and perform the necessary operations based on the message context and content.

Passing Criteria:

- The BPI service orchestration accurately identifies the actions to be completed based on the message context and content.
- The identified actions align with the desired actions for the given message.
- The BPI service orchestration successfully orchestrates the desired actions.
- All required services are invoked and perform the necessary operations based on the message context and content.
- The overall execution of the BPI service orchestration completes without any errors or exceptions.
- The expected outcome or result of the orchestrated actions is achieved.

[R88]

BPI service orchestration utilized in a BPI MUST NOT introduce additional points of failure.

In the context of this document, a point of failure is defined as any non-redundant part of a BPI that, if not operational, would cause a BPI to fail.

[R88](#) Testability:

Preconditions:

- A BPI system is operational.
- The BPI service orchestration is properly configured and deployed.
- One or more sets of BPI test transactions.

Test Steps:

1. Execute a set of BPI test transactions.
2. Monitor the behavior of the BPI service orchestration during the execution of each test transaction.
3. Verify that the BPI service orchestration behaves as expected for test transaction.
4. Simulate potential failure scenarios by deliberately disabling or interrupting specific components or services in the BPI.
5. Verify that the BPI service orchestration properly handles the failures and gracefully recovers without causing a complete BPI failure.
6. Simulate potential failure scenarios by deliberately disabling or interrupting the BPI service orchestration in the BPI.
7. Verify that BPI service orchestration failure is discovered and that its built-in redundancies are successfully executed.
8. Monitor the overall stability and resilience of the BPI system during the failure scenarios.

Passing Criteria:

- The BPI service orchestration successfully coordinates the required services and operations during normal operations.
- All critical components and services of the BPI continue to operate as expected throughout the test execution.
- The BPI service orchestration demonstrates resilience and fault tolerance by handling potential failure scenarios without causing a complete BPI failure of other BPI system components.
- The BPI service orchestration demonstrates resilience and fault tolerance by handling potential failure scenarios without causing a complete BPI failure of the BPI service orchestration component.
- The BPI system maintains its overall stability and functionality even in the presence of failures or disruptions.
- No single non-redundant part of the BPI, including the BPI service orchestration, is identified as a point of failure during the test.

[R89]

BPI service orchestration utilized in a BPI MUST have source consistency preservation.

This means that the content of a message/service request cannot be altered after it has been created by the BPI service orchestration capability.

[R89](#) Testability:

Preconditions:

- A BPI system is operational.
- The BPI service orchestration capability is properly configured and deployed.

Test Steps:

Non-Standards Track Work Product

1. Send a service request to the BPI service orchestration capability.
2. Capture the content of the service request before it is processed by the BPI service orchestration.
3. Verify that the content of the service request remains unchanged throughout the processing by the BPI service orchestration.
4. Monitor the BPI service orchestration to ensure that it does not alter the content of the service request during any stage of processing.
5. Repeat steps 1-4 with multiple service requests, covering different scenarios and message types.
6. Perform a comparison between the original content of the service requests and the content received after processing by the BPI service orchestration.
7. Validate that there are no differences or alterations in the content of the service requests.

Passing Criteria:

- The content of the service requests remains unchanged throughout the processing by the BPI service orchestration.
- The BPI service orchestration does not introduce any modifications, additions, or omissions to the content of the service requests.
- The comparison between the original content and the content after processing shows no differences or alterations.

[R90]

BPI service orchestration utilized in a BPI MUST avoid having the orchestration consumers see partial and/or inconsistent data.

Note that transaction boundaries, i.e., a single service's action, may trigger atomic updates.

[R90](#) Testability:

Preconditions:

- A BPI system is operational.
- The BPI service orchestration capability is properly configured and deployed.
- The BPI has data sources or services that provide data to the orchestration process.

Test Steps:

1. Trigger a service request that involves data retrieval or modification through the BPI service orchestration.
2. Monitor the orchestration process to ensure that all required data sources or services are accessed.
3. Verify that the data retrieved or modified by the orchestration process is complete and consistent.
4. Introduce a failure or inconsistency in one of the data sources or services during the orchestration process.
5. Repeat the same service request and monitor the orchestration process.
6. Validate that the orchestration process detects the failure or inconsistency in the data source or service.
7. Ensure that the orchestration process handles the failure or inconsistency appropriately, such as by providing an error message or fallback mechanism.
8. Confirm that the orchestration consumers do not receive partial or inconsistent data during the service request.

Passing Criteria:

1. The BPI service orchestration successfully retrieves and modifies the required data from all relevant data sources or services.
2. In the event of a failure or inconsistency in a data source or service, the orchestration process detects and handles it appropriately.
3. The orchestration consumers do not receive partial or inconsistent data during the service request.
4. The BPI service orchestration ensures that the data seen by the orchestration consumers is complete and consistent, even in the presence of potential failures or inconsistencies in the underlying data sources or services.

[R91]

BPI service orchestration utilized in a BPI MUST capture the exact order in which operations happened.

In the context of this document, operations in a BPI service orchestration do not necessarily have to be causally connected to be ordered by the BPI service orchestration. For example, if operation 1, the creation of an invoice between Alice and Bob in a BPI, is received by the BPI service orchestration of a BPI before operation 2, the creation of a shipping notice of a purchase order between Bob and Claire, then the operation identifiers assigned to operation 1 and operation 2 must reflect that order through for example a deterministic nonce as the operation's identifier.

[R91](#) Testability:

Preconditions:

- a BPI system is operational.
- The BPI service orchestration capability is properly configured and deployed.
- The BPI has multiple operations or services that are orchestrated.

Test Steps:

1. Perform a series of operations or service requests through the BPI service orchestration.
2. Monitor the orchestration process to ensure that it captures the order of the operations.
3. Validate that the orchestration process maintains an accurate and sequential record of the operations.
4. Introduce variations in the timing or order of the operations.
5. Repeat the operations and monitor the orchestration process.
6. Verify that the orchestration process accurately captures the adjusted order of the operations.
7. Perform operations with concurrent requests from multiple users or systems.
8. Monitor the orchestration process to confirm that it correctly records the order of operations from different sources.
9. Test the orchestration process with a large number of operations to ensure scalability and accurate ordering.
10. Validate that the captured order of operations matches the actual order in which they were performed.

Passing Criteria:

- The BPI service orchestration accurately captures the order of operations or service requests.
- The orchestration process maintains the exact order of operations even with variations in timing or concurrent requests.
- The orchestration process can handle a large number of operations without compromising the accuracy of the captured order.
- The captured order of operations matches the actual order in which they were performed, providing a reliable audit trail of the BPI activities.

[R92]

BPI service orchestration utilized in a BPI MUST preserve a consistent state.

[R92](#) Testability:

Preconditions:

- A BPI system is operational.
- The BPI service orchestration capability is properly configured and deployed.
- The BPI has multiple operations or services that are orchestrated.

Test Steps:

Non-Standards Track Work Product

1. Execute a series of operations or service requests through the BPI service orchestration.
2. Monitor the BPI service orchestration state changes and data modifications during the orchestration process.
3. Verify that the BPI service orchestration state remains consistent throughout the orchestration, with no unexpected changes or inconsistencies.
4. Introduce variations or errors in the input data or requests.
5. Repeat the operations and monitor the state changes.
6. Validate that the orchestration process handles variations and errors without compromising the consistency of the state through proper error handling.
7. Perform concurrent operations from multiple users or systems.
8. Monitor the state BPI service orchestration changes during the concurrent operations.
9. Ensure that the state remains consistent and does not lead to conflicts or data corruption.

Passing Criteria:

- The BPI service orchestration maintains a consistent state throughout the execution of operations.
- The orchestration process handles variations and errors without introducing inconsistencies or corruption in the state.
- Concurrent operations do not result in conflicts or data corruption, preserving the consistent state of the BPI.

[R93]

BPI service orchestration utilized in a BPI MUST ensure that a service subscriber is decoupled from a BPI service it invokes.

This ensures that there is less load and more stability of the orchestration stack and that subscribers are decoupled from the state machine, such that they can independently scale.

R93 Testability:

Preconditions:

- A BPI system is operational.
- The BPI service orchestration capability is properly configured and deployed.
- There are subscribers to the BPI service.

Test Steps:

1. A BPI service subscriber invokes a BPI service subscribed by multiple subscribers.
2. Monitor the execution of the BPI service operation.
3. Verify that the service request invokes a BPI service instance independent of the invoking subscriber.
4. Validate that only the invoking subscriber receives the expected responses or notifications from the BPI service invocation.
5. Modify the system state and trigger another invocation of the BPI service.
6. Verify that the service request invokes a BPI service instance independent of the invoking subscriber.
7. Validate that only the invoking subscriber receives the expected responses or notifications from the BPI service invocation.

Passing Criteria:

1. The BPI service orchestration ensures that the service request invokes BPI service instance independent of the invoking subscriber
2. Only the invoking subscriber receives the expected responses or notifications from the BPI service invocation.
3. The BPI system maintains its operational functionality throughout the service invocations.
4. The subscribers maintain their functionality and responsiveness to the BPI service without being directly involved in the execution.

[R94]

BPI service orchestration utilized in a BPI MUST NOT make assumptions about consumer uptime.

R94 Testability:

Preconditions:

- A BPI system is operational.
- The BPI service orchestration capability is properly configured and deployed.

Test Steps:

1. Bring down one or more consumers of the BPI service.
2. Invoke the BPI service that relies on the consumer(s) that are currently offline.
3. Monitor the behavior of the BPI service orchestration.
4. Verify that the BPI service orchestration maintains the service requests for the offline consumers.
5. Bring the offline consumer(s) back online.
6. Observe how the BPI service orchestration handles the availability of the consumer(s).
7. Validate that the BPI service is able to seamlessly interact with the consumer(s) without any negative impact on the service orchestration.

Passing Criteria:

1. The BPI service orchestration maintains the state of service requests when consumer(s) are offline.
2. The BPI service is able to handle the availability of the consumer(s) gracefully and resumes normal operation when they come back online.
3. No data or functionality is lost or compromised due to the offline status of the consumer(s).
4. The BPI system maintains its overall functionality and responsiveness throughout the test.

[R95]

BPI service orchestration utilized in a BPI MUST isolate data sources from data consumers.

R95 Testability:

Preconditions:

1. The BPI system is operational.
2. The BPI service orchestration capability is properly configured and deployed.

Test Steps:

1. A BPI service subscriber invokes a BPI service that requires access to data from a specific BPI data source.
2. Monitor the data flow and interactions between the BPI service and the data source.
3. Verify that the data consumer, the BPI service, accesses the data source using the BPI service orchestration, and not directly.
4. Repeat steps 1. through 3. for concurrent BPI service invocations.
5. Ensure that each invoked BPI service receives the necessary data from the data source through the BPI service orchestration independent from any of the other BPI service requests.

Passing Criteria:

1. The BPI service accesses the data source through the BPI service orchestration, and not directly.
2. Each invoked BPI service receives the necessary data from the data source through the BPI service orchestration independent from any of the other BPI service requests.

Non-Standards Track Work Product

[R96]

BPI service orchestration MUST not have a Material Impact on the overall system latency of the BPI.

[R96] Testability:

Preconditions:

- A BPI test system with a BPI service orchestration capability is set up and running.
- A test BPI service subscriber and BPI service consumer are set up within the BPI.
- The BPI has a defined threshold for "Material Impact".

Test Steps:

1. The BPI service subscriber invokes a BPI service.
2. Measure the time it takes for the BPI service request to propagate through the BPI service orchestration component and reach the BPI service consumer.
3. Compare the measured time with the defined "Material Impact" threshold.

Test Passing Criteria:

- The test is considered passed if the measured time for the BPI service request to propagate through the BPI service orchestration component and reach the BPI service consumer is below the defined "Material Impact" threshold.

[R97]

BPI service orchestration utilized in a BPI MUST NOT have a negative Material Impact on BPI scalability and BPI availability.

This requirement ensures that overall system latency is not impacted when volume meaningfully and rapidly changes (scalable) at any given point in time (highly available).

[R97] Testability:

Preconditions:

- A BPI test system is set up and running with BPI service orchestration implemented.
- There are enough resources available to simulate varying volume loads on the system.
- Define minimum and maximum processing volume for the test.
- Define acceptable BPI service orchestration latencies for the test.
- Define BPI system latencies for the test.

Test Steps:

1. Increase the volume of data being processed by the BPI service orchestration component to a significant level.
2. Verify that the overall system latency is within acceptable limits.
3. Gradually increase the volume of data being processed until it reaches a peak level.
4. Verify that the overall system latency is still within acceptable limits.
5. Reduce the volume of data being processed back to the initial level.
6. Verify that the overall system latency returns to its previous level.

Passing Criteria:

- In steps 2 and 4, the overall system latency should not exceed the acceptable limit specified for the BPI.
- In step 6, the overall system latency should return to its previous level.
- The test should be repeated multiple times with varying volume loads, and the passing criteria should be met consistently.

5.4 BPI Communication

The BPI Communication capability is a foundational element of any BPI to facilitate workflows in workgroups. Therefore, it is supposed to meet:

- the workflow communication and workflow orchestration requirements between BPI Subjects, and
- the highly asynchronous nature of BPI transactions originating from such workflows.

This leads to the following core capability requirements within a BPI:

[R98]

BPI communication protocols MUST be message-based.

[R98] Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI communication capability is enabled and configured to use message-based protocols.

Test Steps:

1. Send a test message from one BPI Subject to another BPI Subject using the BPI communication capability.
2. Verify that the test message is delivered successfully to the intended recipient BPI Subject.
3. Repeat steps 1 and 2 for multiple test messages to ensure consistent message delivery.
4. Attempt to send a test message using a non-message-based protocol (e.g., direct API call).
5. Verify that the BPI communication capability does not process the test message sent via the non-message-based protocol.

Passing Criteria:

- All test messages sent via the BPI communication capability are successfully delivered to the intended recipients.
- Test messages sent using non-message-based protocols are not processed by the BPI communication capability.

[R99]

BPI communication protocols MUST be asynchronous.

[R99] Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI communication capability is enabled and configured to use asynchronous communication protocols.

Non-Standards Track Work Product

Test Steps:

1. Send a message from BPI Subject A to BPI Subject B using the BPI communication capability.
2. Verify that BPI Subject A can continue with other operations without waiting for a response from BPI Subject B.
3. BPI Subject B should receive the message and process it asynchronously without blocking other operations.
4. Send multiple messages from BPI Subject A to BPI Subject B in rapid succession.
5. Observe that BPI Subject A can send all messages without waiting for individual responses.
6. Ensure that BPI Subject B processes all messages independently and asynchronously.
7. Attempt to send a response message from BPI Subject B to BPI Subject A using the same communication channel.
8. Verify that BPI Subject A can receive the response message asynchronously without any delays in other operations.
9. Repeat steps 1 to 8 with different message sizes and content to validate asynchronous behavior under various conditions.
10. Initiate a long-running task in BPI Subject B, such as a computational operation or a time-consuming operation.
11. Verify that BPI Subject A can continue to send messages to BPI Subject B during the execution of the long-running task.
12. Monitor BPI Subject B to ensure that the long-running task does not affect the processing of incoming messages.
13. Introduce a delay or network latency to simulate slow message transmission between BPI Subject A and BPI Subject B.
14. Send a message from BPI Subject A to BPI Subject B during the simulated delay.
15. Verify that BPI Subject A does not wait for the response and can proceed with other operations.
16. Ensure that BPI Subject B receives the message and processes it asynchronously once the delay is resolved.

Passing Criteria:

- Test messages sent via the BPI communication capability are processed asynchronously by both the sender and receiver.
- BPI Subject A can continue with other operations without waiting for responses from BPI Subject B.
- BPI Subject B processes all incoming messages independently and asynchronously, even during long-running tasks or network latency.
- Asynchronous behavior is consistent under different message sizes, content, and network conditions.
- The BPI system demonstrates reliable and consistent asynchronous communication throughout the test.

[R100]

BPI communication protocols **MUST** be simplex.

Simplex is a communication mode in which only one message is transmitted, and always going in the same direction.

An example of the three requirements above is as follows: BPI Subject X sends a message over channel A at time t_0 to BPI Subject Y. It may receive a response from BPI Subject Y over channel B at a later time T_1 .

[R100] Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI communication capability is enabled and configured to use simplex communication protocols.

Test Steps:

1. Send a message from BPI Subject A to BPI Subject B using the BPI communication capability.
2. Verify that BPI Subject B receives the message successfully.
3. Attempt to send a response message from BPI Subject B to BPI Subject A using the same communication channel.
4. Verify that the response message is not transmitted, as simplex communication only allows messages to flow in one direction.
5. Repeat steps 1 to 4 with multiple test messages to ensure consistent simplex behavior.
6. Send multiple messages from BPI Subject A to BPI Subject B in rapid succession.
7. Observe that each message is transmitted separately and unidirectionally, without any overlapping or simultaneous transmission.
8. Attempt to initiate multiple simultaneous message transmissions from BPI Subject B to BPI Subject A.
9. Verify that only one message is transmitted at a time, and subsequent messages wait for the ongoing transmission to complete.

Passing Criteria:

- Test messages sent via the BPI communication capability are successfully transmitted in one direction and received by the intended recipients.
- Response messages are not allowed in simplex communication, and any attempted response transmissions are blocked.
- Simultaneous message transmissions are not possible in simplex mode, and messages are transmitted sequentially without overlapping.

[R101]

BPI communication protocols **MUST** be based on established communication protocol standards.

Non-normative examples include, but are not limited to, NATS [\[NATS\]](#), AMQP [\[AMQP\]](#), and DIDComm [\[DIDCOMM\]](#).

Note that typically, and in the context of this document, a communication protocol encompasses four layers:

- *Semantic layer*
- *Routing layer*
- *Cryptographic layer*
- *Transport layer*

[R101] Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI communication capability is enabled and supports various communication protocols.

Test Steps:

1. Identify the list of communication protocols that the BPI claims to support based on the documentation.
2. For each communication protocol, verify that it adheres to well-established communication protocol standards (e.g., HTTP, HTTPS, MQTT, WebSockets, AMQP, etc.).
3. Confirm that the communication protocols used in the BPI are widely accepted and have been widely used in the industry.
4. Check if the BPI follows the defined specifications and requirements for each supported communication protocol.
5. Review the communication protocol documentation to ensure that it covers all necessary aspects, such as message format, headers, payload, authentication, security, and error handling.
6. Verify that the BPI communication protocols use standard message formats and data encoding methods.
7. Validate that the BPI communication protocols support common security mechanisms like TLS/SSL encryption for secure communication.
8. Ensure that the communication protocols are interoperable and can communicate with other systems using the same standard protocols.
9. Review the implementation of each communication protocol to ensure it aligns with the established protocol standards.
10. Send test messages using each communication protocol and examine the responses received.
11. Check if the responses adhere to the standard communication protocol guidelines and have the expected format and structure.

Passing Criteria:

Non-Standards Track Work Product

- All BPI communication protocols used are based on well-established and widely accepted communication protocol standards.
- The BPI adheres to the defined specifications and requirements for each supported communication protocol.
- Communication protocols used by the BPI are interoperable and can communicate with other systems using the same standard protocols.
- Test messages sent through each communication protocol result in responses that adhere to the standard protocol guidelines and have the expected format and structure.
- The BPI communication protocols provide the necessary security features, such as encryption and authentication, as specified by the established communication protocol standards.
- The BPI communication capability is consistent with the established communication protocol standards throughout the test.

The communication protocol layers are defined as follows, and note that some requirements refer to communication between BPIs as noted within the relevant requirement:

• Transport Layer

◦ [\[R102\]](#)

A BPI MUST utilize the well established TLS 1.2 [RFC5246](#) or 1.3 [RFC8446](#) protocol to transport messages between BPIs.

[\[R102\]](#) Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI communication capability is enabled and supports TLS protocol versions 1.2 and/or 1.3.

Test Steps:

1. Check the BPI documentation to confirm that it specifies the use of TLS protocol versions 1.2 and/or 1.3 for message transport between BPIs.
2. Ensure that the BPI supports TLS protocol version negotiation to enable the use of either TLS 1.2 or TLS 1.3 depending on the capabilities of the communicating parties.
3. Review the BPI implementation to verify that the TLS handshake process is properly configured to negotiate the highest supported TLS version with the remote BPI.
4. Validate that the BPI enforces strong cryptographic cipher suites and secure settings in the TLS configuration for both TLS 1.2 and TLS 1.3.
5. Confirm that the BPI uses valid and trusted X.509 certificates for TLS encryption and that certificate validation is performed during the handshake process.
6. Test the BPI communication by initiating TLS connections between BPI instances, ensuring that the selected TLS version adheres to the requirements specified in the documentation.
7. Monitor the TLS handshake process to verify that it successfully negotiates either TLS 1.2 or TLS 1.3 based on the supported versions of the communicating BPIs.
8. Capture and analyze network traffic to ensure that TLS encryption is applied to the message transport between BPIs.
9. Perform a vulnerability assessment to identify potential security issues in the TLS implementation and configuration.

Passing Criteria:

- The BPI documentation clearly states that TLS protocol versions 1.2 and/or 1.3 are utilized for message transport between BPIs.
- The BPI supports TLS protocol version negotiation and can dynamically select the highest supported TLS version during the handshake process.
- The TLS handshake process successfully negotiates either TLS 1.2 or TLS 1.3, depending on the capabilities of the communicating BPIs.
- Strong cryptographic cipher suites and secure settings are enforced for both TLS 1.2 and TLS 1.3 connections in the BPI configuration.
- The BPI uses valid and trusted X.509 certificates for TLS encryption, and certificate validation is performed during the handshake process.
- During testing, the TLS-encrypted connections are established successfully between BPI instances using either TLS 1.2 or TLS 1.3 as negotiated.
- The network traffic analysis confirms that message transport between BPIs is protected by TLS encryption.
- The vulnerability assessment does not identify any critical security issues in the TLS implementation and configuration of the BPI.

◦ [\[R103\]](#)

For synchronous BPI messaging between BPIs, a BPI MUST utilize HTTPS [RFC2818](#).

[\[R103\]](#) Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI communication capability is enabled and supports HTTPS based on RFC 2818.

Test Steps:

1. Check the BPI documentation to confirm that it specifies the use of HTTPS based on RFC 2818 for synchronous messaging between BPIs.
2. Validate that the BPI supports the HTTPS protocol with proper configuration options, including support for secure cipher suites, server authentication, and client certificate validation.
3. Ensure that the BPI enforces the use of HTTPS when communicating with other BPI instances synchronously.
4. Verify that the BPI correctly handles HTTPS URL construction, including specifying the appropriate HTTP method (e.g., GET, POST, PUT, DELETE) and parameters for synchronous communication.
5. Test the BPI communication by initiating synchronous HTTPS connections between BPI instances, ensuring that the expected HTTP status codes (e.g., 200 OK) are received for successful responses and appropriate error codes (e.g., 4xx, 5xx) are received for unsuccessful responses.
6. Capture and analyze network traffic to ensure that HTTPS encryption is applied to the synchronous message exchange between BPIs.
7. Check that the BPI handles SSL/TLS certificate validation correctly and rejects invalid or self-signed certificates.
8. Perform a vulnerability assessment to identify potential security issues related to HTTPS implementation and configuration.

Passing Criteria:

- The BPI documentation clearly states the use of HTTPS based on RFC 2818 for synchronous messaging between BPIs.
- The BPI supports HTTPS with the appropriate configuration options for secure cipher suites, server authentication, and client certificate validation.
- The BPI consistently uses HTTPS for synchronous communication with other BPI instances.
- During testing, the synchronous HTTPS connections are successfully established between BPI instances, and the expected HTTP status codes are received for various test scenarios.
- The network traffic analysis confirms that the synchronous message exchange between BPIs is protected by HTTPS encryption.
- The BPI correctly handles SSL/TLS certificate validation and rejects invalid or self-signed certificates.
- The vulnerability assessment does not identify any critical security issues related to the HTTPS implementation and configuration of the BPI.

◦ [\[R104\]](#)

For asynchronous BPI messaging between BPIs or messaging within a BPI, a BPI MUST utilize established asynchronous protocols such as Websockets or AMQP.

[\[R104\]](#) Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI communication capability is enabled and supports asynchronous messaging.
- The BPI supports one or more established asynchronous protocols such as Websockets or AMQP.

Test Steps:

1. Check the BPI documentation to confirm that it specifies the use of established asynchronous protocols (e.g., Websockets, AMQP) for asynchronous BPI messaging between BPIs or messaging within a BPI.

Non-Standards Track Work Product

2. Verify that the BPI has the required libraries or modules installed to support the chosen asynchronous protocols.
3. Test the BPI's ability to establish asynchronous connections using the chosen protocol. For example:
 - a. For Websockets: - Initiate a connection to the BPI's Websockets endpoint. - Exchange messages between BPI instances asynchronously through Websockets.
 - b. For AMQP: - Connect to the BPI's AMQP broker. - Publish messages to a queue and consume messages asynchronously from the queue.
4. Validate that the BPI can handle concurrent asynchronous message processing and can scale effectively to handle a large number of asynchronous messages.
5. Simulate various real-world scenarios where BPI instances communicate asynchronously, such as handling multiple incoming messages concurrently, and verify that the BPI functions correctly without message loss or misordering.
6. Ensure that the BPI gracefully handles connection failures and reconnects to the asynchronous messaging broker without data loss.
7. Test the BPI's error handling capabilities for asynchronous communication, such as handling message processing failures and reporting errors appropriately.

Passing Criteria:

- The BPI documentation clearly states the use of established asynchronous protocols (e.g., Websockets, AMQP) for asynchronous BPI messaging between BPIs or messaging within a BPI.
- The BPI successfully establishes asynchronous connections using the specified protocols.
- The BPI can effectively exchange messages asynchronously with other BPI instances or within the same BPI using the chosen protocols.
- The BPI handles concurrent asynchronous message processing and scales effectively to handle a large number of asynchronous messages without performance degradation.
- The BPI functions correctly in various real-world scenarios, ensuring proper message handling without loss or misordering of messages.
- The BPI gracefully handles connection failures and reconnects to the asynchronous messaging broker without data loss.
- The BPI demonstrates robust error handling for asynchronous communication, ensuring appropriate handling of message processing failures and error reporting.

Note that while messaging within a BPI must be asynchronous, the communication between two BPIs can be either synchronous to asynchronous depending on the use case.

- Cryptographic Layer: This layer deals with the BPI message envelope and the BPI message payload authenticity. However, it does not deal with authorization. Authorization is assumed to be validated based on security policies in the BPI core components such as workgroups.

◦ [\[R105\]](#)

All BPI envelope level formats MUST be achieved through JOSE-based data structures see [\[R106\]](#) and [\[R107\]](#).

Note that JOSE stands for JSON Object Signing and Encryption.

[\[R105\]](#) Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI supports data exchange using JOSE-based data structures.

Test Steps:

1. Test the BPI's ability to send and receive messages using JOSE-based data structures such as JSON Web Encryption (JWE) and JSON Web Signature (JWS). In particular:
 - a. Send a sample message with JOSE-based data structures as the envelope format to a recipient BPI instance.
 - b. Verify that the recipient BPI can successfully receive and process the message.
2. Perform interoperability tests with other BPI instances or systems that also use JOSE-based data structures to ensure seamless data exchange.
3. Validate that the BPI's error handling mechanisms are in place and provide meaningful error messages when encountering invalid or unsupported JOSE-based data structures.
4. Verify that the BPI enforces security measures defined by JOSE to protect the confidentiality, integrity, and authenticity of the exchanged data.

Passing Criteria:

1. The BPI successfully sends and receives messages using JOSE-based data structures such as JWE or JWS.
2. Interoperability tests with other BPI instances or systems that use JOSE-based data structures demonstrate successful data exchange.
3. The BPI provides meaningful error messages when encountering invalid or unsupported JOSE-based data structures.
4. The BPI enforces security measures defined by JOSE to protect the confidentiality, integrity, and authenticity of the exchanged data.

◦ [\[R106\]](#)

BPI encrypted message formats MUST use an Encrypted JSON Web Encryption (JWE) structure [RFC7516](#).

[\[R106\]](#) Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI supports encryption of messages using JSON Web Encryption (JWE) structure as specified in RFC7516.

Test Steps:

1. Test the BPI's ability to encrypt and decrypt messages using the JWE structure. For example:
 - a. Create a sample JSON payload.
 - b. Encrypt the payload using JWE structure.
 - c. Verify that the encrypted payload is in compliance with RFC7516.
 - d. Decrypt the encrypted payload and ensure it matches the original JSON payload.
2. Verify that the BPI can properly handle various JWE encryption algorithms and key management methods as defined in RFC7518.
3. Perform interoperability tests with other BPI instances or systems that also use JWE structure for message encryption to ensure seamless data exchange.
4. Validate that the BPI enforces the necessary security measures, such as key management and encryption algorithm selection, when using JWE structure.

Passing Criteria:

- The BPI successfully encrypts and decrypts messages using JWE structure, with the decrypted payload matching the original JSON payload.
- The BPI handles various JWE encryption algorithms and key management methods as defined in RFC7518 correctly.
- Interoperability tests with other BPI instances or systems using JWE structure demonstrate successful data exchange.
- The BPI enforces necessary security measures for key management and encryption algorithm selection when using JWE structure.

◦ [\[R107\]](#)

BPI signed unencrypted formats MUST use a Signed JSON Web Signature (JWS) structure following [RFC7519](#) and following [RFC7515](#) for its digital signature.

[\[R107\]](#) Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI supports signing unencrypted formats using JSON Web Signature (JWS) structure as specified in RFC7515 and JWT format as specified in RFC7519.

Test Steps:

1. Test the BPI's ability to generate and verify digital signatures using JWS structure. For example:

Non-Standards Track Work Product

- a. Create a sample JSON payload.
 - b. Sign the payload using JWS structure.
 - c. Verify that the generated signature is in compliance with RFC7515.
 - d. Verify the signature against the original JSON payload to ensure its validity.
2. Verify that the BPI can handle various JWS signing algorithms and key management methods as defined in RFC7518.
3. Perform interoperability tests with other BPI instances or systems that also use JWS structure for digital signatures to ensure seamless data exchange.
4. Validate that the BPI enforces the necessary security measures, such as key management and signature algorithm selection, when using JWS structure for signing.

Passing Criteria:

- The BPI successfully generates and verifies digital signatures using JWS structure, with the verified signature matching the original JSON payload.
- The BPI handles various JWS signing algorithms and key management methods as defined in RFC7518 correctly.
- Interoperability tests with other BPI instances or systems using JWS structure for digital signatures demonstrate successful data exchange.
- The BPI enforces necessary security measures for key management and signature algorithm selection when using JWS structure for signing.

◦ [D20]

BPI messages SHOULD always use JWEs with the ciphertext containing a signed payload.

A JWS may be used for initial authentication and authorization between BPIs.

[D20] Testability:

Preconditions:

- The BPI system is operational and properly configured.
- The BPI supports the use of JSON Web Encryption (JWE) with signed payloads.

Test Steps:

1. Create a sample BPI message payload in JSON format.
2. Encrypt the payload using JWEs with the ciphertext containing a signed payload.
3. Attempt to decrypt the JWE-encrypted payload and verify that the decrypted content matches the original payload.
4. Generate an invalid or tampered JWE by modifying the signed payload or changing the cryptographic parameters.
5. Attempt to decrypt the tampered JWE and verify that the BPI correctly detects the tampering and rejects the message.
6. Test the BPI's behavior when receiving BPI messages that use JWEs with signed payloads from other BPI instances. Verify that the BPI can properly process and decrypt the received messages.

Passing Criteria:

- The BPI successfully encrypts and decrypts BPI message payloads using JWEs with signed payloads, with the decrypted content matching the original payload.
- The BPI correctly detects and rejects invalid or tampered JWEs during decryption.
- The BPI can properly process and decrypt BPI messages received from other BPI instances that use JWEs with signed payloads.

◦ [D21]

The digital signature used for the JWS and JWE of a BPI Message SHOULD be based on the public keys associated with a W3C DID in the W3C DID document as defined in [W3C DID](#).

This simplifies the authentication of the message without having to rely on a 3rd party identity provider to validate the digital certificate issued to the BPI Subject. Non-normative examples of W3C DID Documents can be found in [W3C DID](#).

[D21] Testability:

Preconditions:

- The BPI system is operational and supports the use of JSON Web Signature (JWS) and JSON Web Encryption (JWE).
- The BPI is capable of working with W3C Decentralized Identifiers (DIDs) and the associated DID documents.
- The BPI supports the use of public key cryptography for digital signatures.

Test Steps:

1. Create a sample BPI Message.
2. Generate a W3C DID document that contains the public keys to be used for digital signatures.
3. Sign the BPI Message using the private key corresponding to one of the public keys in the W3C DID document.
4. Attach the JWS and/or JWE to the BPI Message.
5. Transmit the BPI Message to another BPI instance.
6. Verify that the recipient BPI instance can successfully validate the digital signature using the corresponding public key from the W3C DID document that is resolved from the sender's W3C DID.
7. Attempt to verify the digital signature using a different public key not present in the W3C DID document, and ensure that the verification fails.
8. Repeat the test with different BPI Messages and different sets of public keys from the W3C DID document.

Passing Criteria:

- The BPI successfully validates the digital signature of a BPI Message when using the corresponding public key from the W3C DID document.
- The BPI rejects the verification of a digital signature when using a public key not present in the W3C DID document.

◦ [R108]

BPI Message authenticity and proof of control of the private keys MUST be established through a cryptographic challenge-response scheme utilizing a shared secret and the public keys of the involved BPI Subject.

[R108] Testability:

Preconditions:

- The BPI supports cryptographic challenge-response schemes.
- The BPI is capable of generating and managing shared secrets for BPI Subjects.
- The BPI can access and utilize the public keys of the involved BPI Subjects.

Test Steps:

1. Generate a shared secret between two BPI Subjects (BPI Subject A and BPI Subject B).
2. BPI Subject A initiates a cryptographic challenge by sending a challenge message to BPI Subject B.
3. The challenge message should contain the shared secret, a random nonce, and any additional required data.
4. BPI Subject B receives the challenge message from BPI Subject A.
5. BPI Subject B processes the challenge and generates a response by signing the concatenated nonce, shared secret, and additional data using its private key.
6. BPI Subject B sends the response message back to BPI Subject A.

Non-Standards Track Work Product

7. BPI Subject A verifies the response by using BPI Subject B's public key to check the signature.
8. Ensure that the verification is successful and the response is valid.
9. Repeat the test with BPI Subject B initiating the challenge, and BPI Subject A responding to it.

Passing Criteria:

- The BPI can successfully generate and manage shared secrets for BPI Subjects.
- The BPI correctly accesses and utilizes the public keys of the involved BPI Subjects.
- BPI Subject A can initiate a challenge and successfully verify the response from BPI Subject B.
- BPI Subject B can initiate a challenge and successfully verify the response from BPI Subject A.
- The cryptographic challenge-response scheme ensures the authenticity of BPI Messages and provides proof of control of the private keys of the involved BPI Subjects.

An example of a challenge-response system is given in the figure below.

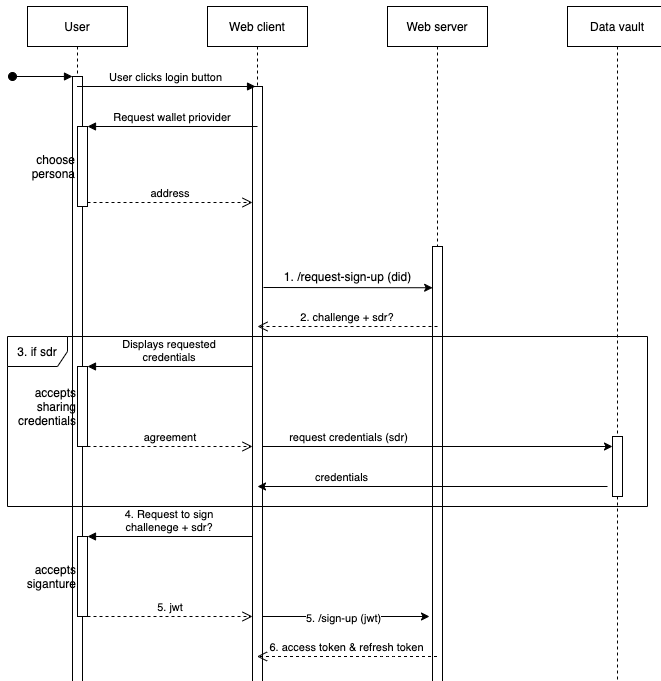


Figure 7: Example of a cryptographic Challenge-Response system between a User, Web Client, Web Server, and a Data Vault based on the DIDAuth protocol using a selective disclosure response (sdr) to the challenge; Source: RSKSmart at <https://rsksmart.github.io/rif-identity-docs/ssi/specs/did-auth.html>

- Routing Layer: A BPI is a "gated community", hence, all BPI capabilities and BPI Subject services are known, or at least directly discoverable, and, therefore, directly addressable within a BPI. Therefore, there is no unknown path between sender and receiver, and, thus, routing is not required as it would be in a public network with an unknown number of participants of unknown identity. The scenario of a BPI Subject with an unknown communication endpoint will be addressed in Section 5.5 BPI Integration.

◦ [R109]

Any BPI Capability addressable using a BPI Message MUST be discoverable by a BPI Subject utilizing resolvable URIs of said BPI capabilities.

An example of such a URI would be a BPI API endpoint

[R109] Testability:

Preconditions:

- The BPI is operational and capable of responding to messages.
- All BPI capabilities are properly configured and addressable through resolvable URIs.

Test Steps:

1. BPI Subject A sends a BPI Message requesting the list of all available BPI capabilities.
2. The BPI receives the message and processes the request.
3. The BPI responds to BPI Subject A with a message containing a list of resolvable URIs for all available BPI capabilities.
4. BPI Subject A parses the response message to extract the list of capability URIs.
5. BPI Subject A selects a specific capability from the list and sends a new BPI Message containing a query for that capability.
6. The BPI receives the message, processes the query, and identifies the specific capability associated with the requested URI.
7. The BPI generates a response message containing the details and parameters of the identified capability.
8. BPI Subject A receives the response message and verifies that the capability details match the expected specifications.
9. BPI Subject A performs the same test for multiple other capability URIs from the initial list.

Passing Criteria:

- The BPI successfully responds to the request for a list of available BPI capabilities.
- The list of capability URIs provided by the BPI is accurate and complete.
- BPI Subject A can select a specific capability from the list and send a query for that capability.
- The BPI correctly identifies the capability associated with the queried URI.
- The BPI provides an accurate and complete response to the query message.
- BPI Subject A can successfully perform the test for multiple capability URIs, and the responses match the expected specifications.

◦ [R110]

Any BPI Subject within a BPI MUST be discoverable by any other BPI Subject within a BPI.

Non-Standards Track Work Product

For example, a BPI Subject DID can be resolved to its DID document containing a BPI communication endpoint in its "service" section that is directly addressable through the BPI Communication capability.

[R110] Testability:

Preconditions:

- The BPI is operational and capable of handling messages.
- All BPI Subjects are registered and accessible within the BPI.

Test Steps:

1. BPI Subject A sends a BPI Message requesting the list of all available BPI Subjects within the BPI.
2. The BPI receives the message and processes the request.
3. The BPI responds to BPI Subject A with a message containing a list of all registered BPI Subjects within the BPI.
4. BPI Subject A parses the response message to extract the list of BPI Subjects.
5. BPI Subject A selects a specific BPI Subject from the list and sends a new BPI Message containing a query for that BPI Subject.
6. The BPI receives the message, processes the query, and identifies the specific BPI Subject associated with the requested information.
7. The BPI generates a response message containing the details and information of the identified BPI Subject.
8. BPI Subject A receives the response message and verifies that the BPI Subject details match the expected specifications.
9. BPI Subject A performs the same test for multiple other BPI Subjects from the initial list.

Passing Criteria:

- The BPI successfully responds to the request for a list of all registered BPI Subjects.
 - The list of BPI Subjects provided by the BPI is accurate and complete.
 - BPI Subject A can select a specific BPI Subject from the list and send a query for that BPI Subject.
 - The BPI correctly identifies the BPI Subject associated with the queried information.
 - The BPI provides an accurate and complete response to the query message.
 - BPI Subject A can successfully perform the test for multiple BPI Subjects, and the responses match the expected specifications.
- Semantic Layer: The semantic layer specifies how a message payload needs to be structured to be both BPI capability/service and BPI Subject friendly. Note that the content level refers to the content and message inside the message envelope. For the semantic layer, this document specifies how messages are identified and processed.

◦ [R111]

Every BPI message MUST contain a message type that allows the context of the message to be established, the content structure to be verified against the context, and the content to be correctly processed.

Message identification does not merely identify the message. The message type also identifies the associated processing protocol such as the specification for a particular zero-knowledge prover scheme. A processing protocol is essentially a group of related messages that are required to achieve a multi-step business process.

[R111] Testability:

Preconditions:

- The BPI is operational and capable of handling messages.

Test Steps:

1. BPI Subject A creates a BPI Message with specifying the message type.
2. BPI Subject A sends the message to the BPI.
3. The BPI receives the message and attempts to process it.
4. The BPI verifies if the message type is present in the message header.
5. The BPI checks if the message type corresponds to a valid and supported message context.
6. The BPI processes the message based on the established context.
7. Repeat step 1. through 6. without specifying a message type.
8. The BPI must reject a BPI message without a message type and generate an appropriate error response to the sender.

Passing Criteria:

- The BPI successfully receives the BPI Message without any errors or disruptions.
- The BPI identifies that the message type is missing from the message header.
- The BPI generates an appropriate error response indicating the absence of the message type.
- The BPI accepts a BPI Message containing a valid message type.
- The BPI confirms that the provided message type corresponds to a recognized and supported message context.
- The BPI processes the BPI Message correctly based on the established context and content structure.
- BPI Subject A receives a response from the BPI indicating successful processing of the BPI Message.

◦ [R112]

A BPI message MUST contain a unique message identifier that is generated by the sender.

This allows unique identification of the message through its lifecycle.

[R112] Testability:

Preconditions:

- The BPI is operational and capable of handling messages.

Test Steps:

1. BPI Subject A creates a BPI Message and generates a unique message identifier for the message.
2. BPI Subject A sends the message to the BPI.
3. The BPI receives the message and extracts the message identifier from the message header.
4. The BPI checks if the message identifier is unique and has not been previously used for any other message.
5. The BPI stores the message identifier for future reference.
6. BPI Subject B creates another BPI Message and generates a unique message identifier for this new message.
7. BPI Subject B sends the second message to the BPI.
8. The BPI receives the second message and extracts the new message identifier.
9. The BPI verifies if the new message identifier is unique and has not been previously used.
10. The BPI stores the new message identifier for future reference.

Passing Criteria:

- The BPI successfully receives the BPI Messages from both BPI Subject A and B without any errors or disruptions.
- BPI Subject A's generated message identifier is unique and has not been previously used in the BPI.

Non-Standards Track Work Product

- The BPI acknowledges that the message identifier from BPI Subject A is unique and stores it for reference.
- BPI Subject B's generated message identifier is unique and has not been previously used in the BPI.
- The BPI acknowledges that the message identifier from BPI Subject B is unique and stores it for reference.
- The BPI is capable of handling multiple unique message identifiers concurrently without any conflicts or errors.

◦ [D22]

A BPI message SHOULD contain one or more message decorators.

In general, decorators in messages at a content level allow for the support of reusable conventions that are present across multiple messages to handle the same functionality consistently. A relevant analogy for decorators is that they are like HTTP headers in an HTTP request. The same HTTP header is often reused as a convention across multiple requests to achieve cross-domain functionality.

*An initial set of useful message decorators that can be used are, but are not limited to: * ~Thread: provide request/reply and threading semantics to allow for maintaining state within, and also across messages * ~Timing: timestamps, expiration, elapsed time * ~L10n: localization support*

[D22] Testability:

Preconditions:

- The BPI is operational and capable of handling messages.

Test Steps:

1. BPI Subject A creates a BPI Message with one or more message decorators.
2. BPI Subject A sends the message to the BPI.
3. The BPI receives the message and extracts the message decorators from the message header.
4. The BPI validates that the message decorators present in the message are correctly formatted and follow the expected conventions.
5. The BPI processes the message decorators to handle the functionality they represent consistently.
6. BPI Subject B creates another BPI Message with a different set of message decorators.
7. BPI Subject B sends the second message to the BPI.
8. The BPI receives the second message and extracts the message decorators.
9. The BPI verifies that the message decorators in the second message comply with the defined conventions.
10. The BPI processes the message decorators to execute the corresponding cross-domain functionality.

Passing Criteria:

- The BPI successfully receives the BPI Messages from both BPI Subject A and B without any errors or disruptions.
- The message decorators present in the message from BPI Subject A are well-formed and adhere to the expected conventions.
- The BPI processes the message decorators from BPI Subject A as intended and consistently handles the functionality they represent.
- The message decorators in the message from BPI Subject B are correctly formatted and follow the predefined conventions.
- The BPI successfully processes the message decorators from BPI Subject B to execute the corresponding cross-domain functionality.
- The BPI demonstrates the ability to handle various message decorators concurrently and consistently, contributing to enhanced message functionality.

◦ [R113]

All content level BPI messages MUST be represented in JSON format [RFC7159].

[R113] Testability:

Preconditions:

- The BPI is operational and capable of processing messages.
- The BPI has defined content level messages that it supports.

Test Steps:

1. BPI Subject A creates a content level BPI Message in JSON format.
2. BPI Subject A sends the JSON-formatted message to the BPI.
3. The BPI receives the message and verifies that it is in JSON format according to RFC7159.
4. The BPI parses the JSON content to extract the relevant data and context of the message.
5. BPI Subject B creates another content level BPI Message in a different format (non-JSON).
6. BPI Subject B sends the non-JSON message to the BPI.
7. The BPI receives the non-JSON message and validates that it does not comply with the JSON format as specified in RFC7159.
8. The BPI attempts to parse the non-JSON content but identifies the format mismatch.
9. BPI Subject C creates another content level BPI Message in JSON format and includes nested JSON structures.
10. BPI Subject C sends the JSON-formatted message with nested JSON structures to the BPI.
11. The BPI receives the message and ensures that the nested JSON structures are valid and correctly processed.
12. The BPI successfully extracts data and context from the nested JSON structures.

Passing Criteria:

- The BPI successfully receives the content level BPI Message from BPI Subject A without any errors or disruptions.
- The content level BPI Message from BPI Subject A is verified to be in JSON format according to RFC7159.
- The BPI accurately extracts the relevant data and context from the JSON-formatted message sent by BPI Subject A.
- BPI Subject B's non-JSON content level BPI Message is received by the BPI, and the BPI detects that it does not adhere to the JSON format specified in RFC7159.
- The BPI handles the non-JSON message gracefully, displaying a format mismatch error and preventing unintended parsing.
- BPI Subject C's content level BPI Message with nested JSON structures is received by the BPI without any issues.
- The BPI successfully processes the nested JSON structures within the message, accurately extracting data and context from the nested JSON elements.

◦ [R114]

BPI Messages MUST be either a valid JSON-LD document, or a JSON document that can be interpreted as JSON-LD by associating a context through HTTP headers, as described in the section "Interpreting JSON as JSON-LD" of the JSON-LD standard [JSONLD].

Note that BPI Messages may fully, and directly support JSON-LD.

[R114] Testability:

Preconditions:

- The BPI is operational and capable of processing messages.
- The BPI has defined content level messages that it supports.

Test Steps:

1. BPI Subject A creates a BPI Message as a valid JSON-LD document with a context provided within the JSON-LD structure.
2. BPI Subject A sends the JSON-LD formatted message to the BPI.

Non-Standards Track Work Product

3. The BPI receives the JSON-LD message and validates that it is indeed a valid JSON-LD document.
4. The BPI extracts the data and context from the JSON-LD message based on the provided context within the JSON-LD structure.
5. BPI Subject B creates another BPI Message as a valid JSON document without any explicit JSON-LD context.
6. BPI Subject B sends the JSON-formatted message to the BPI.
7. The BPI receives the JSON message and identifies that it lacks an explicit JSON-LD context.
8. The BPI checks the HTTP headers of the received message for any context association.
9. If the BPI identifies a context in the HTTP headers, it interprets the JSON message as JSON-LD and extracts the data and context accordingly.
10. If the BPI does not find a context in the HTTP headers, it treats the message as a standard JSON document and processes it accordingly.

Passing Criteria:

- The BPI successfully receives the JSON-LD formatted BPI Message from BPI Subject A without any errors or disruptions.
- The BPI validates that the received BPI Message is a valid JSON-LD document.
- The BPI accurately extracts the relevant data and context from the JSON-LD formatted message sent by BPI Subject A.
- BPI Subject B's JSON BPI Message is received by the BPI without any issues.
- The BPI identifies that the JSON BPI Message lacks an explicit JSON-LD context.
- The BPI checks the HTTP headers of the received message and identifies an associated context.
- If a context is found in the HTTP headers, the BPI interprets the JSON BPI Message as JSON-LD and extracts the data and context accordingly.
- If no context is found in the HTTP headers, the BPI treats the message as a standard JSON document and processes it accordingly.

5.5 BPI Integration

BPI Integration or BPI Interoperability, meaning the ability of two or more BPIs to functionally interoperate with one another, is important to generate network effects.

This document defines BPI Interoperability as follows: BPI A and BPI B are said to be interoperable if Alice can advance the state of one or more of her state objects from BPI A by synchronizing said state object(s) with one or more of Bob's state objects on BPI B. The synchronization is achieved by creating a joint state between Alice and Bob without having to exit the state objects of either participant to the underlying CCSM. The joining of states must occur in an, ideally, censorship-resistant manner and such that neither Alice, nor Bob, nor one of their delegates can maliciously alter and finalize the state of the joint state object at any point before, during, and after the state change of the joint state object on either of the participating BPIs or the (permissionless) CCSMs used by either BPIs.

Note, that the above definition refers to "East-West" or horizontal interoperability between BPIs. In other words the ability of Alice on BPI A to functionally interact with Bob on BPI B, such as issuing an invoice. North-South interoperability, in other words, the vertical interoperability of components such as transaction processing or storage within a BPI, is out of scope for the current version of the standard.

A BPI East-West interoperability solution in the context of this document has but is not limited to the following characteristics in no particular order:

- Resolvable, public key controlled identifiers for all participants following established standards such as W3C DIDs to enable portability across BPIs. Resolvable refers to the identifier being used to discover its controlling keys and other control, authorization, and service attributes of an identifier. For example, an Ethereum address can function as a resolvable public key controlled identifier. See for example the specification of an Ethereum based DID method ([did:ethr](#)).
- Discoverable authentication/authorization capabilities based on common, well-established frameworks such as OAuth2 ([DAUTH-2.0](#)), OpenIDConnect ([OIDC](#)), SIOP DID AuthN (OIDC compatible DID Auth) ([SIOP](#)) to avoid reinventing the wheel and also signaling openness to enterprises.
- Discoverable and negotiable services such as Authentication and Authorization endpoints, a price oracle endpoint, etc. In this context, a service has consumers, providers, input and output parameters, and associated business logic that transforms the input into the output parameters. Discoverable in this context means that a Service Consumer can find a Service Provider and what the capabilities of the service are as well as the required input and output parameters. Negotiable in this context means that the Service Consumer and Provider can negotiate how the service is delivered. This is required to be able to automate interoperability processes between BPIs.
- Bi-or Multi-directional and mono-directional services where bi-/multi-directional services in this context refer to direct and either synchronous or asynchronous service-consumer-to-service-provider or vice versa state object synchronizations and advancements via APIs. Mono-directional in this context refers to services that either export state objects from or deliver state objects to a BPI via APIs. Alice and Bob might want to exchange or synchronize state objects across BPIs or just extract and keep their state objects locked in their environment before committing to another BPI.
- Standardized set of APIs such as REST representing common asset functionalities
 - with defined API endpoint functionalities such as transfer, lock, unlock, exit, deliver, swap
 - standardized API envelopes consisting of for example BPI origin and target metadata, security parameters, etc.
 - standardized asset payloads describing the assets and their current state; a current state can consist of for example a (zero-knowledge) proof set (state-object-history, state-object-locks, state-object-state) and state object meta-data such as state-object type, state-object ID, state-object owner, anchor contract(s) for proof verification(s) – See also related work [here](#). It is critical to have the same "words and grammar" to be able to talk to one another.
- Discoverable Standard Transport security such as JOSE with JWS/JWE or DIDComm is critical to ensure security and privacy at all times, and beyond HTTPS. See also related work [here](#).

Before outlining the requirements for BPI Interoperability this document wants to provide a concrete, detailed, and real-world non-normative example of BPI Interoperability.

One of the most common processes occurring every day across the world is the importing of goods into a country, and delivering them to a customer in that country. This involves a shipper of the goods, Alice, an importer of the goods, Bob, and a country customs organization, Claire.

This scenario has, therefore, 3 workflows intersecting which are independent yet related to one another: Alice has her shipping workflow of containers coming into a harbor with goods that Bob with his importing workflow wants to pass through Claire's custom workflow to deliver it to his customer.

Some BPI Workflow Interoperability Prerequisites:

In the context of these 3 BPIs having to interoperate, Alice, Bob, and Claire must provide at least the following:

1. Alice has identified Bob and Claire as required counterparties for the next workstep in Alice's shipping workflow (container customs processing).
2. Neither Bob nor Claire is part of Alice's BPI.
3. Alice, Bob, and Claire have W3C DIDs with communication service endpoints such that they can send and receive messages from one another.
4. Alice, Bob, and Claire have agreed upon a zero-knowledge prover scheme and one or more worksteps expressed through human-readable predicates that allow them to synchronize their respective workflow states and deterministically advance the joint state.
5. Alice has the following data set available from the last workstep of her shipping workflow, called Set A – a ZK proof, associated public input data, proof-verification key, a verification endpoint for the ZK proof verification, and a [Predicate](#) as a human-readable version of the ZK Proof.
6. Bob has the following data set available from the last workstep of his import workflow, called Set B – a ZK proof, associated public input data, proof-verification key, a verification endpoint for the ZK proof verification, and a [Predicate](#) as a human-readable version of the ZK Proof.
7. Claire has the following data set available from the last workstep of her customs workflow, called Set C – a ZK proof, associated public input data, proof-verification key, a verification endpoint for the ZK proof verification, and a [Predicate](#) as a human-readable version of the ZK Proof.

The usage of ZK proofs is required since the counterparties will not be able to see sensitive information that has entered either of the workflows. For example, intelligence service information about a particular load in a shipping container cannot be shared with Alice and Bob, or the identity of the ultimate load recipient cannot be disclosed to Alice by Bob.

Furthermore, and for the simplicity of the example here, it is assumed that the ZK proofs in the workflow for each participant are recursive. This means that a verification of the ZK proof of the last

Non-Standards Track Work Product

workstep in a workflow proves the correctness and integrity of the entire workflow up to the last workstep for which a valid ZK proof was constructed.

Example BPI Interoperability Workflow – See Figure 8 below:

1. Alice locks the state of her current Workflow's Workstep on her BPI
2. Alice opens a state channel committing an initial state consisting of Set A

Note, any singleton state machine such as a state channel, sidechain, or other Layer 2 solution may be employed.

3. Alice sends state channel invites to Bob and Claire based on their communication endpoints in their DID documents, creating the equivalent of a temporary Workgroup. The invite must include the state information both Bob and Claire need to submit to the state channel, Set B and set C, and the required agreement criteria e.g. 2 of 3 digital signatures over the joined state of the ZK proofs in the form of an agreement [Predicate](#). The [Predicate](#) would be a ZK prover circuit with specified private and public inputs that can be run by Bob and Claire to generate ZK proofs proving correct state updates. In this case, simple signatures over the ZK-proofs are chosen.
4. After accepting the invites, Bob and Claire commit their states to the state channel in the following manner -> $\text{newStateRoot} = H(H(\text{Alice state}, \text{Bob State}), H(\text{Claire State}))$ and also store the inputs (Alice state, Bob State, Claire State)
5. Each participant validates each other's proofs on their respective CCSMs (e.g. Alice checks Bob's and Claire's ZK-proofs) utilizing the supplied verification endpoints.
6. Each participant then signs over the other's submitted signatures (e.g. Bob over Alice's submitted signature over her ZK proof/public data) and submits those signatures as state updates into the state channel.

For example, Bob goes first and submits his signatures (Alice State, Bob New State = Bob Old State + Signature over Alice State + Signature of Claire State, Claire State) with the $\text{newStateRoot} = H(H(\text{Alice State}, \text{Bob New State}), H(\text{Claire State}))$, then Claire (Alice State, Bob New State, Claire New State = Claire Old State + Claire Sig over Bob's Sig over Alice's Sig + Claire's Sig over Bob's original Sig) with a new state root, then Alice, and so forth until all signatures are submitted and all original states have been attested to by the counterparties. This creates a new final state of a synchronized state between the three workflows.

7. Alice, Bob, and Claire each extract the final state from the state channel and exit the channel.
8. Alice, Bob, and Claire can now use this (synchronized) new state as an input in the next worksteps in their respective workflows.

Because the state advancement in the state channel is strictly deterministic and dependent on the defined exit criteria, each workflow has now properly synchronized inputs for their next worksteps in their respective workflows.

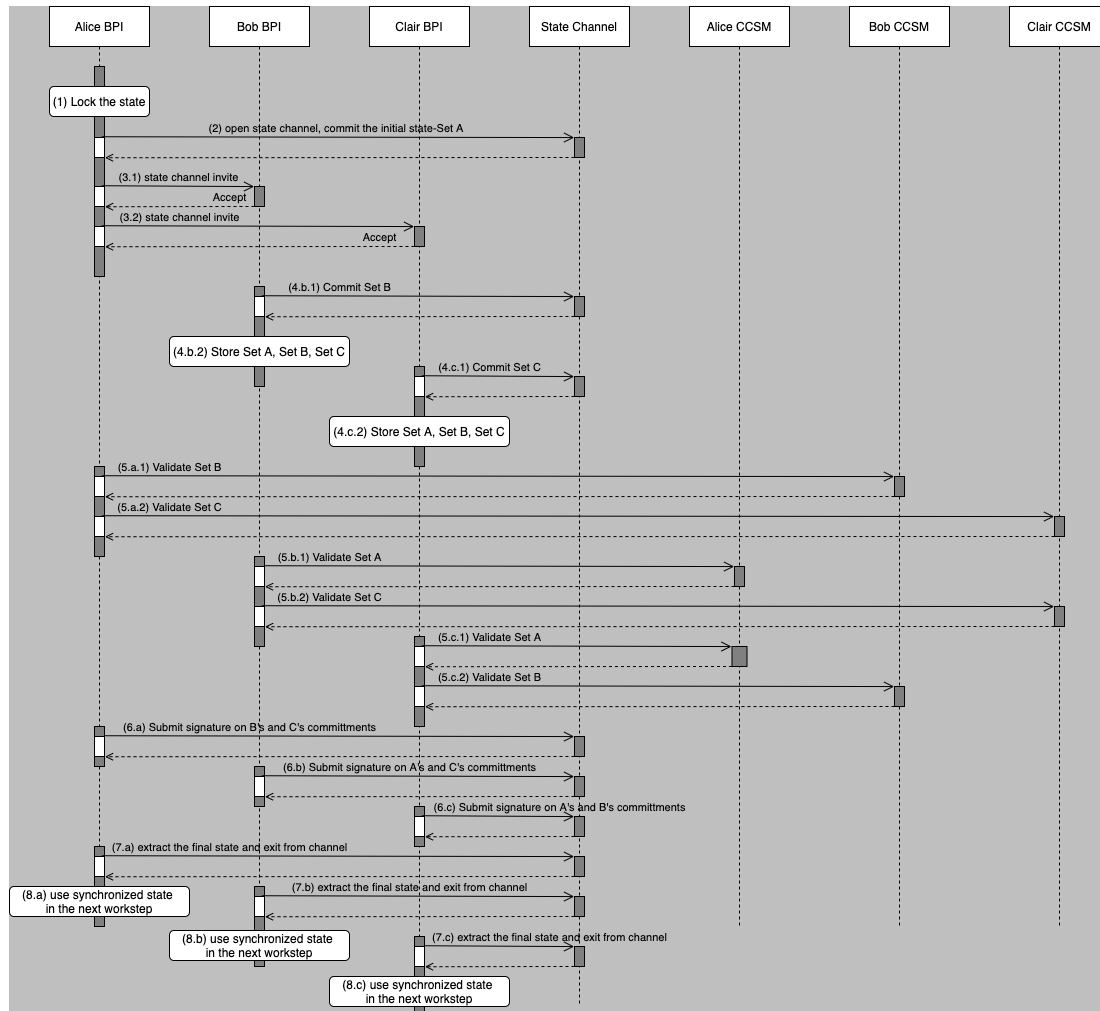


Figure 8: Illustrative example of BPI Interoperability Workflow. Source: Mehran Shakeri (SAP)

5.5.1 Resolvable Identifiers for BPI Interoperability Services

Resolvable identifiers utilized within a BPI are of equivalent importance for interactions between BPIs.

[R115]

Non-Standards Track Work Product

Identifiers and Credentials utilized in BPI Interoperability MUST satisfy all requirements in section 3 [Identifiers, Identity and Credential Management](#).

[R115] Testability: See the Testability Statements in section 3 [Identifiers, Identity and Credential Management](#).

5.5.2 Discoverable Authentication and Authorization Capabilities for BPI Interoperability Services

Similar to a BPI itself, the interactions between BPIs through BPI Interoperability services must follow a similar, albeit slightly reduced set of requirements around BPI Subject authentication and authorization as a BPI.

[R116]

BPI Interoperability Services MUST be compatible with widely used external authentication services.

Non-normative examples of such authentication technologies are OAUTH [\[OAUTH-2.0\]](#) , SAML [\[SAML\]](#) , OIDC [\[OIDC\]](#), AD/LDAP [\[ActiveDirectory\]](#).

[R116] Testability:

Preconditions:

- The BPI is operational and capable of interacting with external services.
- The external authentication services e.g. OAUTH, SAML, OIDC, or AD/LDAP are configured and available.

Test Steps:

1. BPI Subject A initiates an authentication request using the OAUTH protocol to authenticate with the BPI.
2. The BPI receives the authentication request and processes it using the OAUTH protocol.
3. The BPI validates the user's identity using the OAUTH authentication flow.
4. Repeat steps 1. to 3. for other supported authentication protocols

Passing Criteria:

- A BPI Subject successfully authenticates using either of the supported authentication protocols with the BPI.
- The BPI processes the authentication request without errors and follows the chosen protocol.
- The BPI accurately validates the BPI Subject's identity through the authentication flow.

Note: Each authentication step should occur independently and without interference from previous steps.

[R117]

BPI Interoperability Services MUST support roles & access management.

Role and Access Management in this context refers to the required roles of BPI Subjects and their authority to access and execute BPI functionality across multiple BPIs.

[R117] Testability:

Preconditions:

- The BPI is operational and capable of interacting with other BPIs.
- Role and Access Management for BPI Subjects is configured and established.

Test Steps:

1. BPI Subject X attempts to access a specific BPI functionality on BPI A that requires Role A.
2. BPI A receives the access request and validates whether BPI Subject X has Role A.
3. If BPI Subject X has Role A, BPI A grants access and allows the execution of the requested functionality.
4. BPI Subject Y tries to access a certain BPI feature on BPI B that necessitates Role B.
5. BPI B receives the access request and verifies if BPI Subject Y possesses Role B.
6. If BPI Subject Y holds Role B, BPI B permits access and allows the execution of the requested functionality.
7. BPI Subject Z attempts to access a specific BPI capability on BPI C that requires both Role C and Role D.
8. BPI C receives the access request and checks whether BPI Subject Z has both Role C and Role D.
9. If BPI Subject Z possesses both Role C and Role D, BPI C grants access for the requested functionality.
10. BPI Subject W attempts to access a particular BPI functionality on BPI D without the required Role.
11. BPI D receives the access request and confirms that BPI Subject W lacks the necessary Role.
12. BPI D denies access to the requested functionality for BPI Subject W.

Passing Criteria:

- BPI Subject X successfully accesses the BPI functionality on BPI A that requires Role A.
- BPI A processes the access request and correctly verifies the possession of Role A.
- BPI Subject Y effectively accesses the BPI feature on BPI B that necessitates Role B.
- BPI B handles the access request and accurately confirms the presence of Role B.
- BPI Subject Z appropriately accesses the BPI capability on BPI C that requires both Role C and Role D.
- BPI C handles the access request and accurately confirms the presence of both Role C and Role D.
- BPI Subject W is denied access to the BPI functionality on BPI D due to the lack of required Role.

Note: Each access request should be independent, and the roles should be accurately verified according to the established access management configuration.

[R118]

BPI Interoperability Services MUST support policy management.

Policy Management in this context refers to the management of authentication and authorization rules to access and execute BPI functionality for roles of BPI Subjects across multiple BPIs.

[R118] Testability:

Preconditions:

- The BPI is operational and capable of interacting with other BPIs.
- Policy Management for BPI Interoperability Services is configured and operational.

Test Steps:

1. BPI Subject X attempts to access a specific BPI functionality on BPI A.
2. BPI A receives the access request and checks the policy management rules for the requested functionality.
3. If the policy management rules grant access to BPI Subject X, BPI A allows the execution of the requested functionality.
4. BPI Subject Y tries to access a certain BPI feature on BPI B.
5. BPI B receives the access request and verifies the policy management rules for the requested feature.
6. If the policy management rules permit access to BPI Subject Y, BPI B allows the execution of the requested feature.

Non-Standards Track Work Product

7. BPI Subject Z attempts to access a specific BPI capability on BPI C.
8. BPI C receives the access request and validates the policy management rules for the requested capability.
9. If the policy management rules authorize access for BPI Subject Z, BPI C grants access for the requested capability.
10. BPI Subject W attempts to access a particular BPI functionality on BPI D without satisfying the policy management rules.
11. BPI D receives the access request and checks the policy management rules for the requested functionality.
12. BPI D denies access to the requested functionality for BPI Subject W due to policy restrictions.

Passing Criteria:

- BPI Subject X successfully accesses the BPI functionality on BPI A as allowed by the policy management rules.
- BPI A processes the access request and correctly verifies the policy management rules for the requested functionality.
- BPI Subject Y effectively accesses the BPI feature on BPI B as permitted by the policy management rules.
- BPI B handles the access request and accurately validates the policy management rules for the requested feature.
- BPI Subject Z appropriately accesses the BPI capability on BPI C in accordance with the policy management rules.
- BPI C processes the access request and correctly confirms the policy management rules for the requested capability.
- BPI Subject W is denied access to the BPI functionality on BPI D due to policy restrictions.

Note: Each access request should be independent, and policy management rules should be accurately verified according to the established policy configuration.

5.5.3 Discoverable and Negotiable BPI Interoperability Services

The discovery of BPI Interoperability services is facilitated through BPI identifiers representing BPI Operator identities resolvable to BPI Interoperability services URIs of a BPI as described in section [3 Identifiers, Identity and Credential Management](#).

To facilitate ease of discovery, a BPI may publish a verifiable and non-repudiable list of its BPI Subjects. Ideally, each entry consists of W3C verifiable credentials establishing the membership of a BPI Subject in a BPI with credential subjects resolvable to the offered BPI services.

[R119]

The service endpoint specification of the resolvable BPI identifier MUST list BPI service endpoints that allow for further BPI service discovery and the establishment of a secure channel between either BPIs or BPI Subjects.

[R119] Testability:

Preconditions:

- The BPI system is operational and capable of handling service endpoint specifications for resolvable BPI identifiers.

Test Steps:

1. Retrieve the resolvable BPI identifier for BPI A.
2. Examine the service endpoint specification associated with the resolvable BPI identifier of BPI A.
3. Verify that the service endpoint specification provides information enabling further BPI service discovery.
4. Check if the service endpoint specification allows for the establishment of a secure channel between BPI A and other BPIs or BPI Subjects.
5. Retrieve the resolvable BPI identifier for BPI B.
6. Examine the service endpoint specification linked to the resolvable BPI identifier of BPI B.
7. Confirm that the service endpoint specification supports additional BPI service discovery.
8. Ensure that the service endpoint specification facilitates the creation of a secure channel between BPI B and other BPIs or BPI Subjects.
9. Retrieve the resolvable BPI identifier for BPI C.
10. Analyze the service endpoint specification associated with the resolvable BPI identifier of BPI C.
11. Validate that the service endpoint specification includes information for further BPI service discovery.
12. Check if the service endpoint specification enables the establishment of a secure channel between BPI C and other BPIs or BPI Subjects.

Passing Criteria:

- The service endpoint specification of the resolvable BPI identifier for BPI A provides details enabling additional BPI service discovery.
- The service endpoint specification of the resolvable BPI identifier for BPI A allows for the secure channel establishment with other BPIs or BPI Subjects.
- The service endpoint specification of the resolvable BPI identifier for BPI B includes information for further BPI service discovery.
- The service endpoint specification of the resolvable BPI identifier for BPI B permits the creation of a secure channel with other BPIs or BPI Subjects.
- The service endpoint specification of the resolvable BPI identifier for BPI C contains details for additional BPI service discovery.
- The service endpoint specification of the resolvable BPI identifier for BPI C enables the establishment of a secure channel with other BPIs or BPI Subjects.

Note: Each test step verifies the accuracy and functionality of the service endpoint specification associated with a resolvable BPI identifier.

[D23]

Each BPI service SHOULD be defined by an identifier that corresponds to the resolvable BPI identifier and its associated signing key(s) that offer this BPI service.

[D23] Testability:

Test Preconditions:

- The BPI is set up with multiple BPI services and their associated signing keys.
- Resolvable BPI identifiers and their corresponding BPI services are available.

Test Steps:

1. Identify a BPI service within the BPI for testing.
2. Retrieve the identifier and associated signing key(s) for the selected BPI service.
3. Create a BPI message that requests information about the available BPI services.
4. Send the BPI message to the BPI service discovery mechanism.
5. Receive the response containing the list of available BPI services and their identifiers.
6. Verify that the identifier of the selected BPI service is listed in the response.
7. Compare the retrieved signing key(s) of the selected BPI service with the signing key(s) associated with its identifier.
8. Perform digital signature verification using the public key associated with the identifier to ensure that the signing key(s) match.

Passing Criteria:

- The response from the BPI service discovery mechanism contains the identifier of the selected BPI service.
- The retrieved signing key(s) match the signing key(s) associated with the BPI service's identifier.
- The digital signature verification using the public key associated with the identifier is successful.

[D24]

Each BPI service SHOULD be minimally specified by the following elements:

- A BPI service domain such as "BPIStateProcessing"

Non-Standards Track Work Product

- A BPI service description
- A BPI service version,
- The BPI service provider,
- The BPI Service endpoint(s),
- The BPI Service endpoint-specification endpoint(s),
- The Cryptographic Prover system,
- The Cryptographic Prover system specification.

[D24] Testability:

Test Preconditions:

- The BPI is set up with multiple BPI services and their specifications.
- Resolvable BPI identifiers are available.

Test Steps:

1. Choose a specific BPI service from the BPI ecosystem for testing.
2. Retrieve the specified elements for the selected BPI service: domain, description, version, provider, service endpoints, service endpoint-specification endpoints, cryptographic prover system, and cryptographic prover system specification.
3. Create a BPI message requesting information about the specified elements of BPI services.
4. Send the BPI message to the BPI service discovery mechanism.
5. Receive the response containing information about available BPI services and their specifications.
6. Verify that the retrieved information matches the specified elements of the selected BPI service.
7. Cross-reference the service endpoints and endpoint-specification endpoints to ensure accuracy.
8. Check if the cryptographic prover system and its specifications are accurately listed.

Passing Criteria:

- The response from the BPI service discovery mechanism contains accurate information about the specified elements of the selected BPI service.
- The retrieved service endpoints and endpoint-specification endpoints match the ones specified.
- The cryptographic prover system and its specifications are correctly listed.

[R120]

To ensure a BPI service specification timeline, the BPI service specification MUST include "created" and "updated" timestamps, and the full history of "updated" timestamps.

[R120] Testability:

Test Preconditions:

- A BPI service specification with accurate timestamps is available.

Test Steps:

1. Retrieve the BPI service specification for testing.
2. Check if the BPI service specification contains the "created" and "updated" timestamps.
3. Verify the accuracy of the "created" timestamp.
4. Check if there is a history of "updated" timestamps associated with the BPI service specification.
5. Compare the "updated" timestamps history with external records to ensure accuracy.

Passing Criteria:

- The BPI service specification includes both "created" and "updated" timestamps.
- The "created" timestamp is accurate and matches the expected creation time.
- The "updated" timestamps history is present and aligned with external records.

[R121]

After a BPI service endpoint has been discovered by a BPI or BPI Subject, the exact specification of the BPI service endpoint MUST be negotiated between the different BPIs and their BPI Subjects.

Negotiation in the context of this document is the process of two or more BPIs presenting each other with and then agreeing to a way to fulfill BPI service requests amongst each other.

[R121] Testability:

Test Preconditions:

- Two or more BPIs capable of negotiation are available for testing.

Test Steps:

1. Initiate a BPI service request from one BPI to another.
2. Monitor the negotiation process between the two BPIs.
3. Ensure that the negotiation process involves the presentation of methods or ways to fulfill the BPI service request.
4. Confirm that the negotiation process results in an agreement between the BPIs on how to fulfill the BPI service request.

Passing Criteria:

- The negotiation process between the BPIs involves presenting ways to fulfill the BPI service request.
- The BPIs successfully reach an agreement on how to fulfill the BPI service request.

Based upon published BPI service capabilities, a BPI service requester may present the BPI service provider with one or more service requests expressed using one of the published specifications of the requested BPI service. Such requirements may include BPI Service Version, List of BPIs offering a BPI service, etc.

[R122]

A BPI service requester MUST present the BPI service provider with one or more service requirements expressed using one of the published specifications of the requested BPI service.

[R122] Testability:

Test Preconditions:

- Two or more BPIs capable of negotiation are available for testing.

Test Steps:

1. Initiate a BPI service request from one BPI to another.

Non-Standards Track Work Product

2. Monitor the negotiation process between the two BPIs.
3. Ensure that the negotiation process involves the presentation of methods or ways to fulfill the BPI service request.
4. Confirm that the negotiation process results in an agreement between the BPIs on how to fulfill the BPI service request.

Passing Criteria:

- The negotiation process between the BPIs involves presenting ways to fulfill the BPI service request.
- The BPIs successfully reach an agreement on how to fulfill the BPI service request.

[O2]

The service provider MAY respond to one of the requirements.

An example of this could be a JSON object with a response e.g. the standard HTTP response 200 or 403

[O2] Testability:

Test Preconditions:

- A BPI service requester is available for testing.
- A BPI service provider with the requested BPI service specification is available.

Test Steps:

1. Prepare a set of service requirements for the requested BPI service.
2. Present the prepared service requirements to the BPI service provider.
3. The BPI service provider responds to one of the presented service requirements.

Passing Criteria:

- The BPI service requester presents service requirements to the BPI service provider.
- The BPI service provider responds to one of the presented service requirements.
- The response from the BPI service provider aligns with the example of a valid JSON response or a standard HTTP response status code.

[CR15]<[O2]

If the BPI service provider has not responded to any of the BPI service requests by a BPI, the BPI service provider MUST refuse the service request.

[CR15]<[O2] Testability:

Test Preconditions:

- A BPI service requester is available for testing.
- A BPI service provider with the requested BPI service specification is available.

Test Steps:

1. Initiate a BPI service request to the BPI service provider.
2. Monitor the response from the BPI service provider.
3. If the BPI service provider responds to the request, proceed to step 4. If not, proceed to step 5.
4. Confirm that the BPI service provider's response aligns with the expected response for a successful service request.
5. Attempt another BPI service request to the same BPI service provider.
6. Monitor the response from the BPI service provider.
7. If the BPI service provider does not respond, verify that the service request is refused by the BPI service provider.

Passing Criteria:

- The BPI service requester initiates a BPI service request to the BPI service provider.
- The BPI service provider responds to the request or does not respond.
- If the BPI service provider responds, the response aligns with the expected response for a successful service request.
- If the BPI service provider does not respond to subsequent requests, the BPI service provider correctly refuses the service request.

5.5.4 Bi- and Multi-directional and Mono-directional BPI Interoperability Services

Mono-directional BPI services are important in the context of a BPI because they allow a BPI Subject to extract or export state objects, their history, and associated metadata from a BPI and subsequently import them into another BPI. To avoid subsequent altering of a state object within a BPI, the state object needs to be provably locked or immobilized on the BPI it came from before it can be exported, and, if required, subsequently imported to another BPI.

Bi-or-multi-directional BPI services facilitate the synchronization of state objects between BPIs as explained at the beginning of this section, and are, therefore, crucially important to avoid the formation of BPI processing islands which would hamper adoption due to vendor lock-in.

First, the document lists the requirements common to mono-directional BPI services and Bi-/Multi-directional services before specifying requirements unique to each type of service.

Requirements common to mono-directional and bi-/multi-directional BPI Services

[R123]

Each BPI state object utilized in one or more mono-directional and bi-/multi-directional BPI Services MUST have a cryptographic non-interactive zero-knowledge proof of correctness.

More details on cryptographic zero-knowledge proofs of correctness in the context of a BPI are given in section 6 Agreement Execution when discussing worksteps. Also, note that the requirements below are very similar to the ones given in section 6.1 BPI Workstep.

[R123] Testability:

Test Preconditions:

- Operational BPI instances.
- BPI State objects used in mono-/bi- or multi-directional BPI services exist.

Test Steps:

1. Select a BPI state object used in a mono-directional BPI service.
2. Verify that the cryptographic zero-knowledge proof of correctness can be successfully generated using the state object's data.
3. Check that the generated cryptographic zero-knowledge proof can be successfully verified.
4. Select a BPI state object used in a bi-/multi-directional BPI service.
5. Verify that the cryptographic zero-knowledge proof of correctness can be successfully generated using the state object's data.
6. Check that the generated cryptographic zero-knowledge proof can be successfully verified.

Non-Standards Track Work Product

Passing Criteria:

- The cryptographic zero-knowledge proof of correctness can be successfully generated for a BPI state object used in both mono-directional and bi-/multi-directional BPI services.
- The cryptographic zero-knowledge proof of correctness for a state object used in both mono-directional and bi-/multi-directional BPI services can be successfully verified.

[R124]

The non-interactive zero-knowledge proof of correctness of a BPI state object utilized in one or more mono-directional and/or bi-/multi-directional BPI Services MUST be succinct.

Succinct in the context of zero-knowledge proofs means that a zero-knowledge proof is verifiable by any 3rd party in a time that is sublinear to the size of the prover system that generated the proof.

[R124] Testability:

Preconditions:

- A non-interactive zero-knowledge proof of a state object utilized in one or more mono-directional and/or bi-/multi-directional BPI Services from a prover system that can generate a zero-knowledge proof of correctness for the output of size N. The size N is given by the size of the circuit used to generate the proof in terms of the number of circuit constraint equations.
- A verifier system that can verify the zero-knowledge proof.

Test Steps:

1. Verify the zero-knowledge proof using the verifier system.
2. Measure the number of computational steps taken by the verifier system to verify the proof.
3. Verify that the number of computational steps taken to verify the proof is proportional at most to or less than the poly-logarithm of the size of the prover system.

Test Passing Criteria:

- The test will pass if the non-interactive zero-knowledge proof of a state object utilized in one or more mono-directional and/or bi-/multi-directional BPI Services can be verified by the verifier system in a number of computational steps at most proportional to the size of the prover system that generated the proof.

[R125]

The non-interactive zero-knowledge proof of correctness of a BPI state object utilized in one or more mono-directional and/or bi-/multi-directional BPI Services MUST be individually available on the CCSM utilized by the BPI after it has been finalized on the BPI (Liveness).

The zero-knowledge proof of correctness can be a cryptographic aggregator of proofs of correctness that would allow multiple proofs to be represented and provable within one proof.

[R125] Testability:

Test Preconditions:

- BPI State objects with associated non-interactive zero-knowledge proofs are available for testing.
- BPI system and CCSM are operational.

Test Steps:

1. Finalize a BPI state object used in a mono-directional BPI service.
2. Retrieve the non-interactive zero-knowledge proof associated with the finalized BPI state object.
3. Verify that the non-interactive zero-knowledge proof is accessible and retrievable from the CCSM.
4. Finalize a BPI state object used in a bi-/multi-directional BPI service.
5. Retrieve the non-interactive zero-knowledge proof associated with the finalized BPI state object.
6. Verify that the non-interactive zero-knowledge proof is accessible and retrievable from the CCSM.

Passing Criteria:

- The non-interactive zero-knowledge proof associated with finalized BPI state objects used in both mono-directional and bi-/multi-directional BPI services can be individually retrieved from the CCSM.

[R126]

The non-interactive zero-knowledge proof of correctness of a BPI state object utilized in one or more mono-directional and/or bi-/multi-directional BPI Services MUST be verifiable by any 3rd party on the CCSM utilized by the BPI (censorship-resistant individual proof verifiability).

[R126] Testability:

Test Preconditions:

- BPI state object with its associated non-interactive zero-knowledge proof is available for testing.
- BPI system and CCSM are operational.

Test Steps:

1. Retrieve a BPI state object and its associated zero-knowledge proof.
2. Provide the non-interactive zero-knowledge proof to a 3rd party not involved in the BPI process.
3. Instruct the 3rd party to verify the non-interactive zero-knowledge proof using the CCSM.
4. Verify that the 3rd party is able to independently verify the correctness of the BPI state object's non-interactive zero-knowledge proof on the CCSM.

Passing Criteria:

- The 3rd party successfully verifies the zero-knowledge proof of correctness of the BPI state object on the CCSM.

[R127]

The non-interactive zero-knowledge proof of correctness of a BPI state object utilized in one or more mono-directional and/or bi-/multi-directional BPI Services MUST NOT be able to be used in more than one mono-directional and/or bi-/multi-directional BPI Services event at any time.

This requirement is necessary to avoid the usage of the same output as collateral in more than one state-altering event, such as tokenization.

A state object will have to be included in a cryptographic commitment that the state object is locked on the BPI. Note that this only restricts the usage of pledged outputs to the CCSM utilized in a given BPI.

[R127] Testability:

Preconditions:

- A BPI system is in place.
- Non-interactive zero-knowledge proofs of correctness (NIZKPs) are being used.
- BPI State Objects are utilized within mono-directional and/or bi-/multi-directional BPI Services.

Non-Standards Track Work Product

Test Steps:

1. Create a BPI state object within the BPI system.
2. Initiate a mono-directional or bi-/multi-directional BPI Service event that involves the BPI state object.
3. Attempt to use the same non-interactive zero-knowledge proof of correctness of the state object in another mono-directional or bi-/multi-directional BPI Service event simultaneously.

Passing Criteria:

- The BPI system should detect the attempt to reuse the same non-interactive zero-knowledge proof of correctness in another event.
- The BPI system should prevent the usage of the NIZKP in the second event.
- An error message or notification should be generated indicating that the NIZKP is already committed to a previous event.
- The NIZKP should be locked and exclusive to the first event until it's finalized.
- This locking mechanism should prevent the same output from being used as collateral in more than one state-altering event.

[R128]

A BPI MUST lock a state object utilized in one or more mono-directional and/or bi-/multi-directional BPI Services as a succinct non-interactive zero-knowledge proof of the lock commitment (privacy preservation of an output pledged in a commercial value-creation event).

Note that a lock can contain more than one state object.

[R128] Testability:

Preconditions:

- A BPI system is operational.
- Non-interactive zero-knowledge proofs of correctness (NIZKPs) are supported.
- BPI State Objects are used within mono-directional and/or bi-/multi-directional BPI Services.

Test Steps:

1. Create a BPI state object within the BPI system.
2. Initiate a mono-directional or bi-/multi-directional BPI Service event involving the state object.
3. Verify that the BPI system generates a succinct non-interactive zero-knowledge proof (NIZKP) of the lock commitment for the state object.
4. Attempt to use the locked state object in another event or as collateral for another transaction.

Passing Criteria:

- The BPI system should successfully generate a succinct NIZKP of the lock commitment for the state object.
- The NIZKP should mathematically prove the lock commitment without revealing the actual content of the state object.
- When attempting to use the locked state object in another event, the BPI system should detect that it's already locked.
- The BPI system should prevent the usage of the locked state object in any other event or transaction.
- The NIZKP of the lock commitment should preserve the privacy of the pledged output and ensure that it cannot be used in any other context.

[R129]

The BPI state object lock commitment MUST be committed to the CCSM utilized by the BPI (Liveness).

[R129] Testability:

Preconditions:

- A BPI system is operational.
- Non-interactive zero-knowledge proofs of correctness (NIZKPs) are supported.
- BPI State Objects are used within the BPI system.
- A BPI state object has been created and utilized in a BPI Service event.

Test Steps:

1. Initiate a mono-directional or bi-/multi-directional BPI Service event involving a state object.
2. Verify that the BPI system generates a succinct non-interactive zero-knowledge proof (NIZKP) of the lock commitment for the state object.
3. Check that the NIZKP of the lock commitment is successfully committed to the Centralized Clearing and Settlement Mechanism (CCSM) used by the BPI.
4. Attempt to verify the lock commitment by independently calculating the NIZKP using the state object's information and comparing it with the committed value in the CCSM.

Passing Criteria:

- The BPI system should successfully generate a succinct NIZKP of the lock commitment for the state object.
- The NIZKP of the lock commitment should be accurately and securely committed to the CCSM.
- Independently calculating the NIZKP using the state object's information should match the committed value in the CCSM.
- The lock commitment should be publicly verifiable through the CCSM, ensuring liveness and transparency of the state object's lock status.

[R130]

The BPI state object lock commitment MUST be verifiable by any 3rd party on the CCSM utilized by the BPI (censorship-resistant proof verifiability).

[R130] Testability:

Test Preconditions:

- A BPI system is set up and operational.
- The BPI state object lock commitment has been generated and committed to the CCSM.

Test Steps:

1. The test system initiates a request to verify the BPI state object lock commitment.
2. The CCSM retrieves the BPI state object lock commitment.
3. The CCSM provides the necessary cryptographic proofs and information to the test system.
4. The test system verifies the BPI state object lock commitment using the provided cryptographic proofs and information.
5. The test system confirms whether the verification process was successful.

Passing Criteria:

- The CCSM successfully retrieves the BPI state object lock commitment.
- The cryptographic proofs and information provided by the CCSM allow the test system to verify the BPI state object lock commitment.
- The test system confirms successful verification of the BPI state object lock commitment.

[R131]

Non-Standards Track Work Product

The BPI state object lock commitment MUST be updatable.

State objects are normally imported back into a BPI at which point the lock commitment on the CCSM needs to be updated. Also, for scalability reasons, the lock commitment should represent more than one locked asset on a CCSM, while individual lock commitments can remain on a BPI.

[R131] Testability:

Test Preconditions:

- A BPI system is set up and operational.
- The BPI state object lock commitment has been generated and committed to the CCSM.

Test Steps:

1. The test system initiates an update request for the BPI state object lock commitment.
2. The CCSM retrieves the existing BPI state object lock commitment.
3. The test system provides the updated information for the BPI state object lock commitment.
4. The CCSM updates the lock commitment based on the provided information.
5. The test system retrieves the updated BPI state object lock commitment from the CCSM.
6. The test system verifies that the updated lock commitment matches the provided information.

Passing Criteria:

- The CCSM successfully retrieves the existing BPI state object lock commitment.
- The CCSM updates the lock commitment based on the provided information.
- The test system successfully retrieves the updated BPI state object lock commitment.
- The retrieved updated lock commitment matches the provided updated information.

[R132]

The BPI state object lock commitment MUST only be updated by the owners of the BPI state object.

The requirements below will specify public URIs for BPI Interoperability services. Therefore, we have additional requirements for those public URIs.

[R132] Testability:

Test Preconditions:

- A BPI system is set up and operational.
- BPI state objects and their owners are identified and authenticated.

Test Steps:

1. The test system initiates an update request for the BPI state object lock commitment for a specific state object.
2. The CCSM retrieves the existing BPI state object lock commitment.
3. The test system verifies that the requester is the owner of the BPI state object.
4. The test system provides the updated information for the BPI state object lock commitment.
5. The CCSM updates the lock commitment based on the provided information.
6. The test system retrieves the updated BPI state object lock commitment from the CCSM.
7. The test system verifies that the updated lock commitment matches the provided information.
8. Repeat steps 1. through 3. where the requester is not the owner of the BPI state object.
9. Verify that the BPI state object lock commitment is not updated, and the requester receives an appropriate error notification.

Passing Criteria:

- The CCSM successfully retrieves the existing BPI state object lock commitment.
- The requester's ownership of the BPI state object is verified.
- The CCSM updates the lock commitment based on the provided information.
- The test system successfully retrieves the updated BPI state object lock commitment.
- The retrieved updated lock commitment matches the provided updated information.
- If the requester is not the owner of the BPI state object, the operation is refused by the BPI and an appropriate error notification is sent to the requester.

[R133]

A public validation URI for BPI Interoperability services MUST be resolvable to the underlying target resource.

[R133] Testability:

Test Preconditions:

- BPI Interoperability services are deployed and accessible on a BPI.
- Public validation URIs for BPI Interoperability services are configured and available.

Test Steps:

1. The test system initiates a request to the public validation URI for a specific BPI Interoperability service.
2. The DNS resolves the public validation URI to an IP address.
3. The request reaches the resolved IP address.
4. The service at the resolved IP address responds with the appropriate information.
5. The test system verifies that the response matches the expected content and format.

Passing Criteria:

- The DNS successfully resolves the public validation URI to an IP address.
- The request reaches the resolved IP address.
- The service at the resolved IP address responds without errors.
- The response from the service matches the expected content and format.

[D25]

A public validation URI for BPI Interoperability services SHOULD be independent of the originating BPI.

[D25] Testability:

Test Preconditions:

- BPI Interoperability services are deployed and accessible.
- Public validation URIs for BPI Interoperability services are configured and available.

Non-Standards Track Work Product

- Multiple BPIs are operational.

Test Steps:

1. Choose BPI A as the originating BPI.
2. The test system initiates a request to the public validation URI for a specific BPI Interoperability service from BPI A.
3. The DNS resolves the public validation URI to an IP address.
4. The request reaches the resolved IP address.
5. The service at the resolved IP address responds with the appropriate information.
6. Repeat steps 2-5 using a different BPI B as the originating BPI.

Passing Criteria:

- The DNS successfully resolves the public validation URI to an IP address for both BPI A and BPI B.
- The request reaches the resolved IP address for both BPI A and BPI B.
- The service at the resolved IP address responds without errors for both BPI A and BPI B.
- The responses from the service match the expected content and format for both BPI A and BPI B.

[R134]

A system processing the BPI Interoperability service MUST notify the requesting BPI Subject with human-readable reasoning about either success or failure.

[R134] Testability:

Test Preconditions:

- BPI Interoperability services are deployed and accessible.
- A BPI Subject's request for a BPI Interoperability service is prepared for testing.

Test Steps:

1. The test system initiates a request to a specific BPI Interoperability service.
2. The service processes the request.
3. The service generates a response containing human-readable reasoning for either success or failure.
4. The test system receives the response.

Passing Criteria:

- The response received from the service contains human-readable reasoning.
- If the service successfully processed the request, the human-readable reasoning in the response indicates success and provides a clear explanation.
- If the service encounters an error during processing, the human-readable reasoning in the response explains the cause of the failure.

[R135]

All BPI Interoperability services MUST be cryptographically secured and privacy-preserving.

"Cryptographically secured" in this context means that all communications follow a common authentication and authorization framework, as previously discussed. Privacy-preserving in this context means that all communications are end-to-end encrypted independent of the security properties of the transportation layer.

[R135] Testability:

Test Preconditions:

- BPI Interoperability services are deployed and accessible.
- Cryptographic keys and protocols for authentication, authorization, and end-to-end encryption are established and available.

Test Steps:

- The test system initiates a request to a specific BPI Interoperability service.
- The service processes the request.
- The service uses the established cryptographic mechanisms to authenticate and authorize the request.
- The service encrypts the response using end-to-end encryption.
- The test system receives the encrypted response.
- The test system decrypts the response using the appropriate cryptographic keys.

Passing Criteria:

- The service successfully authenticates and authorizes the request.
- The response received by the test system is encrypted using end-to-end encryption.
- The test system is able to successfully decrypt the encrypted response using the appropriate cryptographic keys.
- The privacy of the communication is preserved throughout the process.

5.5.4.1 Mono-directional BPI services

Mono-directional BPI services in the context of BPI interoperability, not regular BPI transactions as specified in section [6.5 BPI Transactions](#) and section [6.6 BPI Transaction Lifecycle](#), need only to perform two operations – export and import. These operations have to encompass cryptographic material and URIs that allow independent verification of the cryptographic material presented to 3rd parties such as an auditor or another BPI.

[R136]

[Mono-directional BPI services](#) in the context of BPI interoperability MUST support at least two operations – export and import.

[R136] Testability:

Preconditions:

- The BPI system is properly configured and operational.
- Mono-directional BPI services are in place and running.
- The BPI system has data to be transferred, both for export and import.

Test Steps:

1. Access the BPI system and initiate the "export" operation.
2. Access the BPI system and initiate the "import" operation.

Expected Result:

1. The export operation starts without errors and retrieves data from the BPI system. The exported data is in a proper format and contains the expected information.

Non-Standards Track Work Product

2. The import operation starts without errors and successfully inserts data into the BPI system. The imported data is stored correctly and matches the data provided for import.

[R137]

In the context of [BPI interoperability](#), the BPI export operation MUST provide at least the following elements to the invoking BPI Subject:

- The State Object
- Zero-Knowledge Proof(s) of Correctness of the state object and its history
- All public input data to the Zero-Knowledge Proof(s) of Correctness of the state object and its history required to validate the proofs
- Verification Keys for the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- Specification of the prover system of the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- A validation URI of the originating BPI that allows a 3rd party to independently verify the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- A lock commitment of the current state object
- The public input data to the lock commitment
- The Verification Keys for the lock commitment
- Specification of the prover system of the lock commitment
- A validation URI of the originating BPI that allows a 3rd party to independently verify the lock commitment

[R137] Testability:

Preconditions:

- Two BPI systems are set up: one to export the data and one to import the data.
- Both BPI systems are properly configured for interoperability.
- The data in the exporting BPI is in a valid and verifiable state.
- The BPI export operation is accessible and functional.

Test Steps:

1. Initiate the BPI export operation from the exporting BPI system and review the exported data.

Expected Result:

1. The State Object is included in the exported data.
2. The Zero-Knowledge Proof(s) of Correctness are part of the exported data.
3. All public input data for validating the Zero-Knowledge Proofs is provided.
4. The Verification Keys for the Zero-Knowledge Proofs are accessible.
5. The prover system specifications for Zero-Knowledge Proofs are provided.
6. The validation URI for third-party verification is included.
7. The lock commitment of the current state object is part of the export.
8. Public input data for the lock commitment is included.
9. The Verification Keys for the lock commitment are accessible.
10. The prover system specifications for the lock commitment are provided.
11. The validation URI for third-party verification of the lock commitment is included.

[R138]

In the context of BPI interoperability, the BPI import operation MUST provide at least the following elements by the invoking BPI Subject to the target BPI:

- The State Object
- Zero-Knowledge Proof(s) of Correctness of the state object and its history
- All public input data to the Zero-Knowledge Proof(s) of Correctness of the state object and its history required to validate the proofs
- Verification Keys for the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- Specification of the prover system of the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- A public validation URI of the originating BPI that allows a 3rd party to independently verify the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- A lock commitment of the current state object
- The public input data to the lock commitment
- The Verification Keys for the lock commitment
- Specification of the prover system of the lock commitment
- A public validation URI of the originating BPI that allows a 3rd party to independently verify the lock commitment

[R138] Testability:

Preconditions:

- Two BPI systems are set up: one to export the data and one to import the data.
- Both BPI systems are properly configured for interoperability.
- The BPI Subject initiating the import operation has access to the required data and elements.

Test Steps:

1. Initiate the BPI import operation from the importing BPI system.

Expected Result:

1. The State Object is included in the imported data.
2. The Zero-Knowledge Proof(s) of Correctness are part of the imported data.
3. All public input data for validating the Zero-Knowledge Proofs is provided.
4. The Verification Keys for the Zero-Knowledge Proofs are accessible.
5. The prover system specifications for Zero-Knowledge Proofs are provided.
6. The validation URI for third-party verification is included.
7. The lock commitment of the current state object is part of the import.
8. The Verification Keys for the lock commitment are accessible.
9. The prover system specifications for the lock commitment are provided.
10. The validation URI for third-party verification of the lock commitment is included.

5.5.4.2 Bi- or Multi-directional BPI services

Bi- and Multi-directional BPI services in the context of BPI Interoperability enable dynamic processes between BPIs.

In the following, the standard introduces the concept of a [State Synchronization and Advancement Predicate](#) for BPI Interoperability (processes). A [Predicate](#) in the context of this document is understood as an assertion that may be true or false, depending on the values of the variables that occur in it and the logical, well-formed connections between those variables. A [State Synchronization and Advancement Predicate](#) is a definition of an Interoperability Virtual State Machine ([IVSM](#)) based on a set of agreed-upon business rules and business data that is deterministic. It synchronizes and advances the state of committed state objects of participants in the BPI Interoperability process. An [IVSM](#) is an implementation of a [State Synchronization and Advancement Predicate](#). One can think of an [IVSM](#) as a stripped-down version of a BPI with a single workgroup that can process only one workstep.

[R139]

Non-Standards Track Work Product

Bi- or Multi-directional BPI services in the context of BPI interoperability MUST support at least the following operations:

- Create [State Synchronization and Advancement Predicate](#)
- Update [State Synchronization and Advancement Predicate](#)
- Launch [IVSM](#)
- Remove [IVSM](#)
- Commit State
- Invite Participants to BPI Interoperability Process
- Accept/Reject Invite
- Add/Remove BPI Subject
- Verify State
- Verify Lock Commitment
- Update State
- Accept/Reject State Update
- Exit BPI Interoperability

In the different sections below, this document defines the requirements for each operation. Each section will have

- Prerequisites
- Required data properties of the operations
- Validity requirements of an operation's data properties
- Operation execution requirements

[R139] Testability:

Preconditions:

- Two or more BPI systems are set up for bi- or multi-directional BPI services.
- All BPI systems involved are configured for interoperability.
- The BPI Subjects and Participants are properly registered in their respective systems.

Test Steps:

1. Create a [State Synchronization and Advancement Predicate](#) from one BPI system.
2. Update the [State Synchronization and Advancement Predicate](#) from another BPI system.
3. Launch an [IVSM](#) from one of the BPI systems.
4. Remove the [IVSM](#) from the system where it was launched.
5. Commit the State in one of the BPI systems.
6. Invite Participants to the BPI Interoperability Process from one of the BPI systems.
7. Accept or Reject the invite from the Participant side.
8. Add a BPI Subject to the BPI Interoperability from one of the systems.
9. Remove a BPI Subject from the BPI Interoperability from one of the systems.
10. Verify the State from any BPI system.
11. Verify the Lock Commitment from any BPI system.
12. Update the State from any BPI system.
13. Accept or Reject the State Update from the receiving side.
14. Exit the BPI Interoperability from any of the participating systems.

Expected Results:

1. All the above steps are completed without errors or exceptions.
2. Each operation specified in the requirement is supported by the BPI services.
3. The BPI systems synchronize and interoperate correctly during these operations.

Create [State Synchronization and Advancement Predicate](#)

As mentioned above, a [State Synchronization and Advancement Predicate](#) defines and provides an implementation of an enforcement mechanism of the required rules and data to synchronize multiple state objects from different BPIs, and if required, advance this synchronized, joint state to a new joined state. The joint state is subsequently usable in the BPI workflows in the different participating BPIs.

[R140]

A [State Synchronization and Advancement Predicate](#) MUST be a mathematically well-formed, deterministic formula that can be evaluated to true or false as one or more functions of the values of the variables that occur in it.

[R140] Testability:

Preconditions:

- A BPI system with a [State Synchronization and Advancement Predicate](#) capability is set up.

Test Steps:

1. Define the [State Synchronization and Advancement Predicate](#) in the BPI system as a formula with well-formed mathematical syntax.
2. Test the formula with different sets of inputs and repeat each set of inputs multiple times.
3. Provide a set of variables that occur in the formula and evaluate the formula with specific values for these variables.

Expected Results:

1. The [State Synchronization and Advancement Predicate](#) accepts a well-formed mathematical formula.
2. The formula is deterministic, producing the same result for the same input.
3. When the formula is evaluated with specific variable values, it should correctly return "true" or "false" based on the provided input.

[R141]

A [State Synchronization and Advancement Predicate](#) MUST at least contain the following elements:

- A unique identifier for the [Predicate](#) which may be resolvable
- The unique identifier of the BPI Subject within the context of the originating BPI who creates the [Predicate](#)
- A creation date
- An update date
- The content of the [State Synchronization and Advancement Predicate](#) in a human-readable format
- The specification of the rules to synchronize participating BPI state objects
- The specification of the input data to synchronize participating BPI state objects both private and public
- The specification of the output objects of the application of the [State Synchronization and Advancement Predicate](#) to the input data to synchronize participating BPI state objects
- The specification of the rules to advance the state of synchronized, participating BPI state objects
- The specification of the input data to advance the state of synchronized, participating BPI state objects both private and public

Non-Standards Track Work Product

- The specification of the output objects of the application of the [State Synchronization and Advancement Predicate](#) to the input data to advance the state of synchronized, participating BPI state objects
- The specification of the cryptographically verifiable joint state between the participating BPIs and their storage
- The specification of a deterministic program and its runtime environment applying the [State Synchronization and Advancement Predicate](#) rules and data to its input data and generating its output objects
- The specification of the cryptographic prover system utilized in verifying the correct application of the [State Synchronization and Advancement Predicate](#) to its input data
- The specification of the required cryptographic material for the cryptographic prover system
- A digital signature over the [Predicate](#) content tied to a public key associated with the BPI Subject creating the [Predicate](#)

The unique [Predicate](#) identifier allows for the disambiguation of predicates in case a system is processing more than one [Predicate](#) at a time. The BPI Subject identifier assures the assignability of the originator for audit and disambiguation purposes. A human-readable format ensures that in the case of business-sensitive operations both business owners and auditors can understand and analyze the intent of the [Predicate](#). For a system to be able to process the [Predicate](#), the rules as well as input and output data for both state synchronization and advancement, together with the processing program, state storage, and output validation, need to be specified such that they can be implemented in a system through an automated process. The last element ensures [Predicate](#) non-repudiability in case of disputes.

[R141] Testability:

Preconditions:

- The BPI system is configured to accept [State Synchronization and Advancement Predicates](#).
- The BPI Subject creating the [Predicate](#) has a valid public-private key pair.
- The BPI Subject has the necessary permissions to create and update [State Synchronization and Advancement Predicates](#).

Test Steps:

1. Generate a unique identifier for the [Predicate](#) and ensure it is not already in use.
2. Prepare input data for the creation of the [State Synchronization and Advancement Predicate](#), including all the specified elements.
3. Sign the content of the [State Synchronization and Advancement Predicate](#) with the BPI Subject's private key.
4. Submit the [State Synchronization and Advancement Predicate](#) for creation within the BPI system.
5. Retrieve the created [Predicate](#) and check for all of the specified elements.
6. Update the [State Synchronization and Advancement Predicate](#), if necessary, and verify the update process.

Expected Results:

1. The [State Synchronization and Advancement Predicate](#) is successfully created within the BPI system.
2. The created [Predicate](#) contains all the specified elements outlined in the requirement.
3. The digital signature over the [Predicate](#) content is valid and tied to the public key associated with the BPI Subject creating the [Predicate](#).
4. The BPI system correctly associates the created [Predicate](#) with the unique identifier of the BPI Subject.
5. The update to the [State Synchronization and Advancement Predicate](#) is successful, and the updated [Predicate](#) reflects the changes made.

[R142]

The output objects of the application of the [State Synchronization and Advancement Predicate](#) MUST be cryptographic assertions in zero-knowledge that evaluate to either true or false by a verifying party.

[R142] Testability:

Preconditions:

- A BPI system capable of applying the [State Synchronization and Advancement Predicate](#) is set up.
- The [State Synchronization and Advancement Predicate](#) is created and has associated input data.

Test Steps:

1. Apply the [State Synchronization and Advancement Predicate](#) to the input data.
2. Attempt to evaluate these cryptographic assertions in a zero-knowledge context.
3. Attempt to evaluate the cryptographic assertions to ensure they can be assessed as either true or false.

Expected Results:

1. The output objects generated by the [State Synchronization and Advancement Predicate](#) are cryptographic assertions.
2. The cryptographic assertions are structured for zero-knowledge evaluation.
3. The cryptographic assertions can be successfully evaluated by a verifying party as either true or false.

[R143]

For BPI Interoperability, the "Create [State Synchronization and Advancement Predicate](#)" operation a BPI invokes MUST create an object conformant to the requirements [\[R140\]](#) - [\[R142\]](#).

[R143] Testability:

Preconditions:

- A BPI system capable of invoking the "Create [State Synchronization and Advancement Predicate](#)" operation is set up.
- The BPI system has appropriate authorization and permissions to invoke this operation.

Test Steps:

1. Invoke the "Create [State Synchronization and Advancement Predicate](#)" operation within the BPI system.
2. Create an object as a result of this operation. Perform the tests described in [\[R140\]](#) - [\[R142\]](#) on this object.

Expected Results:

1. The "Create [State Synchronization and Advancement Predicate](#)" operation successfully creates an object.
2. The created object conforms to the requirements [\[R140\]](#) - [\[R142\]](#) as specified in the requirement.

[R144]

The input data to the "Create [State Synchronization and Advancement Predicate](#)" operation MUST enable a BPI to generate a [State Synchronization and Advancement Predicate](#) per [\[R141\]](#).

[R144] Testability:

Preconditions:

- A BPI system is set up with the capability to invoke the "Create [State Synchronization and Advancement Predicate](#)" operation.
- The BPI system has the necessary permissions and authorization to execute this operation.

Test Steps:

Non-Standards Track Work Product

1. Invoke the "Create [State Synchronization and Advancement Predicate](#)" operation within the BPI system.
2. Analyze the input data provided to the operation to ensure it aligns with all requirements from [\[R141\]](#). Then, generate the [State Synchronization and Advancement Predicate](#) and perform the tests listed in [\[R141\]](#).

Expected Results:

1. The "Create [State Synchronization and Advancement Predicate](#)" operation successfully executes.
2. The input data for the operation enables the BPI system to generate a [State Synchronization and Advancement Predicate](#) that conforms to the criteria described in [\[R141\]](#).

[\[R145\]](#)

A "Create [State Synchronization and Advancement Predicate](#)" operation MUST satisfy the following conditions to be valid:

- The invoking BPI Subject's digital signature must be valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- A [State Synchronization and Advancement Predicate](#) conformant with [\[R141\]](#).

[\[R145\]](#) Testability:

Preconditions:

- A BPI system is set up and configured to execute the "Create [State Synchronization and Advancement Predicate](#)" operation.
- The BPI system has registered BPI Subjects with their unique identifiers, digital signatures, and associated public keys.

Test Steps:

1. Initiate a "Create [State Synchronization and Advancement Predicate](#)" operation with a valid digital signature, a public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) conformant with [\[R141\]](#).
2. Initiate a "Create [State Synchronization and Advancement Predicate](#)" operation with an invalid digital signature, a public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) conformant with [\[R141\]](#).
3. Initiate a "Create [State Synchronization and Advancement Predicate](#)" operation with a valid digital signature, a public key that is not cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) conformant with [\[R141\]](#).
4. Initiate a "Create [State Synchronization and Advancement Predicate](#)" operation with a valid digital signature, a public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) that is not conformant with [\[R141\]](#).

Expected Results:

1. The "Create [State Synchronization and Advancement Predicate](#)" operation successfully executes.
2. The "Create [State Synchronization and Advancement Predicate](#)" operation fails.
3. The "Create [State Synchronization and Advancement Predicate](#)" operation fails.
4. The "Create [State Synchronization and Advancement Predicate](#)" operation fails.

Update [State Synchronization and Advancement Predicate](#)

In case requirements change, a [State Synchronization and Advancement Predicate](#) may have to be updated. Such an update process needs to be conformant to the following requirements.

[\[R146\]](#)

A [State Synchronization and Advancement Predicate](#) MUST NOT be updated when the [Predicate](#) is used in an active [IVSM](#).

[\[R146\]](#) Testability:

Preconditions:

- An active [IVSM](#) (Interoperable Verifiable State Machine) exists within the BPI.
- The BPI system is configured to use [State Synchronization and Advancement Predicates](#).
- There is at least one [State Synchronization and Advancement Predicate](#) in use within the active [IVSM](#).

Test Steps:

1. Select a specific [State Synchronization and Advancement Predicate](#) that is currently in use within the [IVSM](#) and attempt to initiate an update or modification to the selected [Predicate](#).
2. Monitor and record the BPI system's response to the update request.
3. Repeat the test for each [Predicate](#) in use within the [IVSM](#) if applicable.

Expected Results:

1. The BPI system consistently rejects or prevents the update of a [State Synchronization and Advancement Predicate](#) when it is actively used within an [IVSM](#).

[\[R147\]](#)

An update operation to a [State Synchronization and Advancement Predicate](#) MUST be conformant to [\[R140\]](#) - [\[R142\]](#).

[\[R147\]](#) Testability:

Preconditions:

- A [State Synchronization and Advancement Predicate](#) (Predicate) exists within the BPI.
- The BPI system is configured to allow updates to the [Predicate](#).
- The [Predicate](#) to be updated conforms to the requirements [\[R140\]](#) - [\[R142\]](#).

Test Steps:

1. Prepare the updates to the [Predicate](#), testing if they conform to the requirements [\[R140\]](#) - [\[R142\]](#).
2. Initiate the update operation for the selected [Predicate](#). Then, monitor and record the BPI system's response to the update request.
3. Repeat the test for multiple [Predicates](#), if applicable, to ensure all updates conform to the requirements.

Expected Results:

1. The BPI system accepts the update operation if the [Predicate](#) conforms to requirements [\[R140\]](#) - [\[R142\]](#).
2. The BPI system consistently accepts updates that adhere to the specified requirements and consistently rejects updates that do not conform to the requirements.

[\[R148\]](#)

For BPI Interoperability, the "Update [State Synchronization and Advancement Predicate](#)" operation a BPI invokes MUST create an object conformant to the requirements [\[R140\]](#) - [\[R142\]](#).

[\[R148\]](#) Testability:

Preconditions:

Non-Standards Track Work Product

- A BPI Subject with the authority to update the [State Synchronization and Advancement Predicate](#) (Predicate) exists within the BPI.
- A [Predicate](#) that requires an update is available.
- The BPI system is configured to allow updates to [Predicates](#).

Test Steps:

1. Identify and select a [Predicate](#) that requires an update. Prepare the update operation, ensuring it conforms to the requirements [R140] - [R142].
2. Initiate the "Update [State Synchronization and Advancement Predicate](#)" operation for the selected [Predicate](#). Then, monitor and record the BPI system's response to the update request.
3. Repeat the test for multiple [Predicates](#), if applicable, to ensure all updates conform to the specified requirements.

Expected Results:

1. The update conforms to the requirements [R140] - [R142] and the update operation is prepared.
2. The BPI system accepts the "Update [State Synchronization and Advancement Predicate](#)" operation.
3. The BPI system consistently accepts updates that adhere to the specified requirements and rejects updates that do not conform to the requirements.

[R149]

The input data to the "Update [State Synchronization and Advancement Predicate](#)" operation MUST enable a BPI to generate a [State Synchronization and Advancement Predicate](#) per [\[R141\]](#).

[R149] Testability:

Preconditions:

- A BPI Subject with the authority to update the [State Synchronization and Advancement Predicate](#) (Predicate) exists within the BPI.
- A [Predicate](#) that requires an update is available.
- The BPI system is configured to allow updates to [Predicates](#).

Test Steps:

1. Identify and select a [Predicate](#) that requires an update. Prepare the input data for the "Update [State Synchronization and Advancement Predicate](#)" operation in a format that conforms to the requirements [R141].
2. Initiate the "Update [State Synchronization and Advancement Predicate](#)" operation for the selected [Predicate](#) with the prepared input data. Then, monitor and record the BPI system's response to the update request.
3. Repeat the test for multiple [Predicates](#), if applicable, to ensure all updates conform to the specified requirements.

Expected Results:

1. The input data enables the BPI to generate a [State Synchronization and Advancement Predicate](#) that conforms to the requirements [R141].
2. The BPI system accepts the "Update [State Synchronization and Advancement Predicate](#)" operation with the provided input data.
3. The BPI system consistently generates [Predicates](#) that meet the specified requirements using different sets of input data and accepts the update request when the input data adheres to the specified format and requirements.

[R150]

An "Update [State Synchronization and Advancement Predicate](#)" operation MUST satisfy the following conditions to be valid:

- The invoking BPI Subject's digital signature must be valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- A [State Synchronization and Advancement Predicate](#) conformant with [\[R141\]](#).

[R150] Testability:

Preconditions:

- A BPI Subject with the authority to update the [State Synchronization and Advancement Predicate](#) (Predicate) exists within the BPI.
- A [Predicate](#) that requires an update is available.
- The BPI system is configured to allow updates to [Predicates](#).
- The digital signature key pair of the invoking BPI Subject is available and appropriately managed.

Test Steps:

1. Initiate a "Update [State Synchronization and Advancement Predicate](#)" operation with a valid digital signature, a public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) conformant with [\[R141\]](#).
2. Initiate a "Update [State Synchronization and Advancement Predicate](#)" operation with an invalid digital signature, a public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) conformant with [\[R141\]](#).
3. Initiate a "Update [State Synchronization and Advancement Predicate](#)" operation with a valid digital signature, a public key that is not cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) conformant with [\[R141\]](#).
4. Initiate a "Update [State Synchronization and Advancement Predicate](#)" operation with a valid digital signature, a public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, and a [Predicate](#) that is not conformant with [\[R141\]](#).

Expected Results:

1. The "Update [State Synchronization and Advancement Predicate](#)" operation successfully executes.
2. The "Update [State Synchronization and Advancement Predicate](#)" operation fails.
3. The "Update [State Synchronization and Advancement Predicate](#)" operation fails.
4. The "Update [State Synchronization and Advancement Predicate](#)" operation fails.

Launch [IVSM](#)

As mentioned above, the [IVSM](#) represents an implementation of the [State Synchronization and Advancement Predicate](#). To properly implement an [IVSM](#), the standard defines the following requirements.

[R151]

The "Launch [IVSM](#)" operation a BPI invokes MUST contain the following elements:

- A unique identifier for the [IVSM](#)
- The unique identifier of the BPI Subject within the context of the originating BPI invoking the operation
- A creation date
- A [State Synchronization and Advancement Predicate](#)
- A list of BPI Subject unique identifiers authorized to invoke the [IVSM](#) operations.
- A digital signature over the content of the operation input tied to a public key associated with the BPI Subject invoking the operation

[R151] Testability:

Preconditions:

Non-Standards Track Work Product

- The Baseline Protocol Implementation (BPI) system is properly configured and operational.
- The BPI Subject invoking the "Launch [IVSM](#)" operation has the necessary permissions and authentication.
- [IVSM](#)-related configurations are properly set up within the BPI.

Test Steps:

1. Initiate the "Launch [IVSM](#)" operation with a unique identifier for the [IVSM](#), the unique identifier of the invoking BPI Subject, a creation date, a [State Synchronization and Advancement Predicate](#), a list of BPI Subject unique identifiers authorized to invoke the [IVSM](#) operations, a digital signature over the content of the operation input tied to a public key associated with the BPI Subject invoking the operation.
2. Attempt to initiate the "Launch [IVSM](#)" operation with any of the elements from step one missing.

Expected Results:

1. The "Launch [IVSM](#)" operation is successfully initiated.
2. The "Launch [IVSM](#)" operation's initiation fails.

[R152]

A "Launch [IVSM](#)" operation MUST satisfy the following conditions to be valid:

- The BPI Subject's digital signature must be valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The submitted [State Synchronization and Advancement Predicate](#) is conformant to [\[R140\]](#) - [\[R142\]](#).

[R152] Testability:

Preconditions:

- The Baseline Protocol Implementation (BPI) system is properly configured and operational.
- The BPI Subject invoking the "Launch [IVSM](#)" operation has the necessary permissions and authentication.
- [IVSM](#)-related configurations are properly set up within the BPI.

Test Steps:

1. Execute the "Launch [IVSM](#)" operation from the BPI. Provide the required input parameters, including a unique identifier, BPI Subject identifier, creation date [State Synchronization and Advancement Predicate](#), and the list of authorized BPI Subject identifiers. Capture any response or acknowledgment from the BPI system.
2. Retrieve information about the launched [IVSM](#).
3. Verify that the digital signature associated with the [IVSM](#) is valid. Confirm that the digital signature is correctly tied to the public key associated with the BPI Subject invoking the operation.
4. Confirm that the public key used in the digital signature is cryptographically tied to the unique identifier of the invoking BPI Subject. Ensure that the binding between the public key and the BPI Subject's unique identifier is accurately recorded for the launched [IVSM](#).
5. Confirm that the [IVSM](#) contains the specified [State Synchronization and Advancement Predicate](#). Ensure that the [Predicate](#) is correctly associated with the launched [IVSM](#) and conforms to the requirements [\[R140\]](#) - [\[R142\]](#).

Expected Results:

1. "Launch [IVSM](#)" operation can be initiated successfully.
2. Digital signature is valid and correctly tied to the BPI Subject's public key.
3. Public key is cryptographically tied to the unique identifier of the invoking BPI Subject.
4. [State Synchronization and Advancement Predicate](#) is correctly linked to the [IVSM](#) and conforms to [\[R140\]](#) - [\[R142\]](#).

[R153]

An [IVSM](#) MUST implement a [State Synchronization and Advancement Predicate](#) conformant to [\[R140\]](#) - [\[R142\]](#).

[R153] Testability:

Preconditions:

- The [IVSM](#) is properly integrated and operational within the Baseline Protocol Implementation (BPI) system.
- Appropriate configurations for [State Synchronization and Advancement Predicate](#) are set up.
- The [IVSM](#) has been appropriately initialized and associated with a BPI Subject.

Test Steps:

1. Trigger an operation that involves the [IVSM](#), such as initiating an [IVSM](#) process. Retrieve the [State Synchronization and Advancement Predicate](#) associated with the [IVSM](#). Capture any response or acknowledgment from the BPI system.
 2. Inspect the retrieved [State Synchronization and Advancement Predicate](#). Confirm that the [Predicate](#) contains all the required elements specified in [\[R140\]](#) - [\[R142\]](#).
 3. Examine the [IVSM](#) process execution that involves applying the [State Synchronization and Advancement Predicate](#). Verify that the deterministic program specified in the [Predicate](#) is correctly executed. Confirm that the program's runtime environment is accurately configured, and the output objects are generated as expected.
 4. Confirm that the cryptographic prover system specified in the [Predicate](#) is employed during [IVSM](#) operations. Verify the correctness of cryptographic material used for verification. Validate that the joint state between participating BPIs and their storage is cryptographically verifiable.
- Test Passing Criteria:

Expected Results:

1. [IVSM](#) can be invoked successfully, and the [State Synchronization and Advancement Predicate](#) is retrievable.
2. The [Predicate](#) contains all required elements specified in [\[R140\]](#) - [\[R142\]](#).
3. Deterministic program execution results in the correct generation of output objects.
4. Cryptographic prover system is applied, and the joint state is cryptographically verifiable.

[R154]

An [IVSM](#) MUST implement the following BPI Interoperability operations that can be invoked by BPI Subjects conformant to the requirements of said operations:

- Commit State
- Add/Remove BPI Subject
- Verify State
- Verify Lock Commitment
- Update State
- Reject State Update
- Finalize State
- Exit BPI Interoperability

This represents the minimal set of operations required to synchronize or advance joint state objects by an [IVSM](#).

[R154] Testability:

Non-Standards Track Work Product

Preconditions:

- The [IVSM](#) is deployed and operational.
- BPI Subjects are registered and authorized to invoke [IVSM](#) operations.
- Joint state objects are available for synchronization.

Test Steps:

1. Invoke the "Commit State" operation on the [IVSM](#). Monitor the response from the system or any events triggered by the commit operation by checking for success messages, error messages, or events emitted by the smart contract on the blockchain. Query the [IVSM](#) for the current state or inspect relevant data structures or databases where the joint state is stored.
2. Add a new BPI Subject and remove an existing BPI Subject using the respective operations.
3. Trigger the "Verify State" operation on the [IVSM](#). Specify what constitutes a successful verification and what conditions would lead to a failed verification. Monitor the response from the [IVSM](#) or any events triggered by the "Verify State" operation by checking for success messages, error messages, or events emitted by the system indicating the outcome of the verification.
4. Execute the "Verify Lock Commitment" operation on the [IVSM](#). Clearly define the conditions that make a lock commitment valid, including time constraints, cryptographic checks, or any other criteria specified by the implementation. Monitor the response from the [IVSM](#) or any events triggered by the "Verify Lock Commitment" operation. This might include success messages, error messages, or events indicating the outcome of the verification. Query the [IVSM](#) for the validity status of the lock commitment. If the [IVSM](#) provides detailed information about the validation result, inspect these details. Compare the actual validation results with the defined expected results.
5. Initiate the "Update State" operation on the [IVSM](#). Clearly define the conditions that should lead to a successful update of the joint state, including the data format, authorization checks, or any other criteria specified by the implementation. Monitor the response from the [IVSM](#) or any events triggered by the "Update State" operation by checking for success messages, error messages, or events indicating the outcome of the update. Compare the actual state after the update with the defined expected results.
6. Attempt to update the state with incorrect or unauthorized information. Trigger the "Update State" operation on the [IVSM](#) with the intentionally invalid data. Clearly define the conditions under which the [IVSM](#) should reject the state update as invalid including criteria such as data validation checks, authorization constraints, or other business logic requirements. Monitor the response from the [IVSM](#) or any events triggered by the "Update State" operation, including error messages, failure notifications, or events indicating the rejection of the invalid state update. Compare the actual rejection status and messages with the defined expected results.
7. Execute the "Finalize State" operation on the [IVSM](#). Monitor the response from the [IVSM](#) or any events triggered by the "Finalize State" operation including success messages, completion notifications, or events indicating the finalization of the current state.
8. Invoke the "Exit BPI Interoperability" operation on the [IVSM](#). Monitor the response from the [IVSM](#) or any events triggered by the "Exit BPI Interoperability" operation, including success messages, completion notifications, or events indicating the conclusion of the BPI Interoperability process.

Expected Results:

1. The commit operation is properly processed and the joint state is successfully updated.
2. The [IVSM](#) updates the joint state to reflect the addition or removal of BPI Subjects.
3. The joint state is successfully verified, and the result is accurate.
4. The lock commitment is valid, and the [IVSM](#) provides a positive verification result.
5. The joint state is successfully updated.
6. The [IVSM](#) correctly rejects the unauthorized state update.
7. The state is successfully finalized.
8. The [IVSM](#) successfully exits the BPI Interoperability.

[R155]

For BPI Interoperability, a valid "Launch [IVSM](#)" operation a BPI invokes MUST

- Instantiate an operational [IVSM](#) conformant to [\[R153\]](#) and [\[R154\]](#)
- Include the list of BPI Subjects as part of the target [IVSM](#) joint state object
- Commit the initial state of an [IVSM](#) as a valid, succinct, and efficient zero-knowledge proof of correctness of the initial state on the CCSM together with its public input and verification key
- Return a list of target [IVSM](#) endpoints as URIs for the operations listed in [\[R154\]](#)
- Return a cryptographically secured and masked secret for the invoking BPI Subject

In the context of this document, cryptographically secured and masked means that an attacker cannot unmask the secret without the cryptographic material used to secure and mask the secret, such as a cryptographic secret used in a key exchange protocol.

[R155] Testability:

Preconditions:

- The BPI is deployed and operational.
- BPI Subjects are registered and authorized to invoke BPI operations.
- The [IVSM](#) is not currently launched.

Test Steps:

1. Trigger the "Launch [IVSM](#)" operation on the BPI.
2. Retrieve the status of the [IVSM](#) instantiation process.
3. Examine the joint state object of the target [IVSM](#).
4. Retrieve information about the initial state of the [IVSM](#). Select a well-established zero-knowledge proof system that suits your requirements. Clearly define the statement you want to prove without revealing the actual data. During the setup phase of the zero-knowledge proof system, generate public parameters, like a proving key and a verification key. Utilize the proving key and the private input (initial state data) to generate a zero-knowledge proof. Share the public input, the generated zero-knowledge proof, and the verification key. Submit the generated zero-knowledge proof along with the initial state information to the CCSM for commitment. Submit the initial state information to the CCSM for commitment. Inspect the storage or database where the CCSM stores committed states.
5. Retrieve the list of target [IVSM](#) endpoints returned by the "Launch [IVSM](#)" operation.
6. Retrieve the cryptographically secured and masked secret returned by the operation.

Expected Results:

1. The operation is successfully initiated.
2. The [IVSM](#) is operational and conforms to [\[R153\]](#) and [\[R154\]](#).
3. The list of BPI Subjects is included in the joint state object.
4. The commitment is valid, succinct, and efficiently proves the correctness of the initial state. The proof includes the public input and verification key.
5. The list of endpoints is provided as URIs for the operations specified in [\[R154\]](#).
6. The secret is cryptographically secured and masked, ensuring that an attacker cannot unmask it without the required cryptographic material.

[R156]

The valid zero-knowledge proof of correctness of the initial joint state MUST be publicly verifiable on the CCSM the [IVSM](#) utilizes.

[R156] Testability:

Preconditions:

- The [IVSM](#) has been successfully launched using the "Launch [IVSM](#)" operation.

Non-Standards Track Work Product

- The initial joint state of the [IVSM](#) has been committed with a zero-knowledge proof.

Test Steps:

1. Retrieve information about the initial state of the [IVSM](#). Utilize the public input, private input (initial state), and verification key to generate a zero-knowledge proof for the initial state.
2. Submit the generated zero-knowledge proof along with the public input and verification key to the CCSM for verification. Inspect the storage or database where the CCSM stores committed states.

Expected Results:

1. The information required for public verification is accessible.
2. The CCSM provides confirmation or evidence that the zero-knowledge proof for the correctness of the initial joint state has been successfully verified.

Remove [IVSM](#)

Once the [IVSM](#) has met the defined finalization criteria of the joint state it can be stopped, and removed, but not before.

[R157]

An [IVSM](#) MUST NOT be stopped unless the finalization criteria of the joint state have been met.

[R157] Testability:

Preconditions:

1. The [IVSM](#) is successfully instantiated and operational.
2. A zero-knowledge proof for the correctness of the initial joint state has been generated using the public input, private input, and verification key.

Test Steps:

1. Trigger the "Launch [IVSM](#)" operation on the BPI.
2. Retrieve information about the initial joint state of the [IVSM](#).
3. Utilize the public input, private input (initial joint state), and verification key to generate a zero-knowledge proof for the correctness of the initial joint state.
4. Submit the generated zero-knowledge proof along with the public input and verification key to the CCSM for verification.
5. Inspect the storage or database where the CCSM stores committed states.

Expected Results:

1. The operation is successfully initiated, and the [IVSM](#) is instantiated.
2. Relevant information about the initial joint state is obtained, including data to be committed.
3. The zero-knowledge proof is successfully generated.
4. The CCSM processes the request without errors.
5. The CCSM provides confirmation or evidence that the zero-knowledge proof for the correctness of the initial joint state has been successfully verified.

[R158]

An [IVSM](#) MUST NOT be able to be removed until all participants in the BPI Interoperability process have successfully invoked the "Exit BPI Interoperability" operation.

[R158] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to perform the "Remove [IVSM](#)" operation.
- The "Exit BPI Interoperability" operation has been invoked successfully by all participants.

Test Steps:

1. Invoke the "Remove [IVSM](#)" operation without all participants having successfully invoked the "Exit BPI Interoperability" operation.
2. Invoke the "Exit BPI Interoperability" operation successfully for all participants, and then attempt the "Remove [IVSM](#)" operation.
3. Try to perform operations on the [IVSM](#) after all participants have successfully exited BPI Interoperability and the [IVSM](#) has been removed.

Expected Results:

1. The [IVSM](#) rejects the removal operation, indicating that not all participants have exited BPI Interoperability.
2. The [IVSM](#) accepts the removal operation, as all participants have successfully exited BPI Interoperability.
3. All operations on the removed [IVSM](#) are rejected or result in errors.

[R159]

A "Remove [IVSM](#)" operation a BPI invokes MUST contain the following properties:

- The unique identifier for the target [IVSM](#)
- The unique identifier of the invoking BPI Subject
- A digital signature over the content of the operation input tied to a public key associated with the BPI Subject invoking the operation
- The cryptographically secured and masked secret of the invoking BPI Subject

[R159] Testability:

Preconditions:

- The [IVSM](#) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to perform the "Remove [IVSM](#)" operation.

Test Steps:

1. Trigger the "Remove [IVSM](#)" operation with the required properties, including the target [IVSM](#) identifier, invoking BPI Subject identifier, digital signature, and cryptographically secured masked secret.
2. Trigger the "Remove [IVSM](#)" operation with all properties except the target [IVSM](#) identifier.
3. Trigger the "Remove [IVSM](#)" operation with all properties except the invoking BPI Subject identifier.
4. Trigger the "Remove [IVSM](#)" operation with all properties except the digital signature.
5. Trigger the "Remove [IVSM](#)" operation with all properties except the cryptographically secured masked secret.
6. Try to invoke the "Remove [IVSM](#)" operation with an unauthorized BPI Subject.
7. Use the retrieved public key to verify the digital signature over the content of the "Remove [IVSM](#)" operation.

Expected Results:

1. The [IVSM](#) processes the removal operation successfully.

Non-Standards Track Work Product

2. The [IVSM](#) should reject the removal operation, indicating that the target [IVSM](#) identifier is required.
3. The [IVSM](#) should reject the removal operation, indicating that the invoking BPI Subject identifier is required.
4. The [IVSM](#) should reject the removal operation, indicating that the digital signature is required.
5. The [IVSM](#) should reject the removal operation, indicating that the cryptographically secured masked secret is required.
6. The [IVSM](#) should reject the unauthorized removal operation.
7. The [IVSM](#) successfully verifies the digital signature.

[R160]

A "Remove [IVSM](#)" operation MUST satisfy the following conditions to be valid:

- The provided [IVSM](#) identifier matches the identifier of the target [IVSM](#)
- The invoking BPI Subject's digital signature must be valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The invoking BPI Subject must be an authorized BPI Subject for the target [IVSM](#)
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the target [IVSM](#) for that BPI Subject

[R160] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to perform the "Remove [IVSM](#)" operation.

Test Steps:

1. Trigger the "Remove [IVSM](#)" operation with a valid [IVSM](#) identifier that matches the identifier of the target [IVSM](#).
2. Trigger the "Remove [IVSM](#)" operation with an [IVSM](#) identifier that does not match the identifier of the target [IVSM](#).
3. Trigger the "Remove [IVSM](#)" operation with a valid [IVSM](#) identifier but an invalid digital signature.
4. Trigger the "Remove [IVSM](#)" operation with a valid [IVSM](#) identifier and a valid digital signature, but the public key is not cryptographically tied to the unique identifier of the invoking BPI Subject.
5. Trigger the "Remove [IVSM](#)" operation with a valid [IVSM](#) identifier, a valid digital signature, and a tied public key, but from an unauthorized BPI Subject.
6. Trigger the "Remove [IVSM](#)" operation with a valid [IVSM](#) identifier, a valid digital signature, a tied public key, and an authorized BPI Subject, but with a cryptographically secured and masked secret that does not match the one stored in the target [IVSM](#).

Expected Results:

1. The [IVSM](#) accepts the operation.
2. The [IVSM](#) should reject the operation, indicating a mismatch in identifiers.
3. The [IVSM](#) should reject the operation, indicating that the digital signature is not valid.
4. The [IVSM](#) should reject the operation, indicating that the public key is not correctly tied to the BPI Subject's identifier.
5. The [IVSM](#) should reject the operation, indicating that the invoking BPI Subject is not authorized for the target [IVSM](#).
6. The [IVSM](#) should reject the operation, indicating a mismatch in the secured and masked secret.

[R161]

For BPI Interoperability, a valid "Remove [IVSM](#)" operation that a BPI invokes MUST remove the [IVSM](#) as identified by its unique identifier and conformant to [\[R157\]](#) and [\[R158\]](#).

[R161] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to perform the "Remove [IVSM](#)" operation.

Test Steps:

1. Trigger the "Remove [IVSM](#)" operation with the valid unique identifier of the target [IVSM](#).
2. Trigger the "Remove [IVSM](#)" operation with an invalid or non-existent unique identifier.
3. Try to invoke the "Remove [IVSM](#)" operation without proper authorization.
4. Verify that the [IVSM](#) removal is conformant to the requirements specified in [\[R157\]](#) and [\[R158\]](#) by following the tests listed in those testability statements.
5. Examine the system state after the "Remove [IVSM](#)" operation.

Expected Results:

1. The [IVSM](#) is successfully removed, and the system state reflects the removal.
2. The [IVSM](#) should reject the operation, indicating that the provided unique identifier does not match any existing [IVSM](#).
3. The [IVSM](#) should reject the unauthorized removal attempt.
4. The [IVSM](#) removal adheres to the criteria outlined in [\[R157\]](#) and [\[R158\]](#).
5. The [IVSM](#) is no longer present or operational, and the system state accurately reflects the removal.

Commit State

Once an [IVSM](#) is instantiated, a BPI Subject can commit a BPI Interoperability state object to the [IVSM](#) through the "Commit State" operation to start or contribute to the BPI Interoperability state and process.

[R162]

A BPI Interoperability state object utilized in the "Commit State" operation to the [IVSM](#) MUST have the following properties:

- The unique identifier for the [State Synchronization and Advancement Predicate](#) of the state to be committed is based on
- The unique identifier of the BPI Subject within the context of the originating BPI who commits the state
- The cryptographically secured and masked secret of the invoking BPI Subject
- A creation date
- The State Object
- Zero-Knowledge Proof(s) of Correctness of the state object and its history
- All public input data to the Zero-Knowledge Proof(s) of Correctness of the state object and its history required to validate the proofs
- Verification Keys for the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- A public validation URI of the originating BPI that allows a 3rd party to independently verify the Zero-Knowledge Proof(s) of Correctness of the state object and its history
- A lock commitment of the current state object
- The public input data to the lock commitment
- The Verification Keys for the lock commitment
- A public validation URI of the originating BPI that allows a 3rd party to independently verify the lock commitment
- A digital signature over the state content tied to a public key associated with the BPI Subject committing the state

This operation is de-facto equivalent to the Mono-Directional service of BPI Import because it serves the same purpose.

Non-Standards Track Work Product

[R162] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to perform the "Commit State" operation.

Test Steps:

1. Trigger the "Commit State" operation with a BPI Interoperability state object that includes all the required properties specified in [R162].
2. Trigger the "Commit State" operation with a state object missing one or more properties specified in [R162].
3. Check the unique identifier for the [State Synchronization and Advancement Predicate](#) of the state object.
4. Check the unique identifier of the BPI Subject committing the state.
5. Check for the cryptographically secured and masked secret of the invoking BPI Subject.
6. Check that the creation date of the state object is correctly set.
7. Check for the state object itself within the "Commit State" operation.
8. Check for the properties of the Zero-Knowledge Proof(s) of Correctness, such as proofs, public input data, and verification keys.
9. Check for the properties of the lock commitment, such as the commitment itself, public input data, and verification keys.
10. Confirm that the digital signature over the state content is tied to a public key associated with the BPI Subject committing the state.

Expected Results:

1. The [IVSM](#) processes the operation, and the state object is successfully committed.
2. The [IVSM](#) should reject the operation, indicating that the state object is incomplete.
3. The unique identifier is present and matches the expected value.
4. The unique identifier is present and matches the expected value.
5. The secured and masked secret is present and matches the expected value.
6. The creation date is present and reflects the time of the operation.
7. The state object is present and contains the relevant information.
8. All necessary properties for Zero-Knowledge Proofs are present and correct.
9. All necessary properties for the lock commitment are present and correct.
10. The digital signature is present and matches the expected value.
11. The "Commit State" operation is accepted and processed successfully with a state object that includes all the required properties specified.
12. The [IVSM](#) rejects the operation if the state object is missing one or more properties specified in [R162].

[R163]

An [IVSM](#) processing a "Commit State" operation MUST satisfy the following conditions to be valid:

- The submitted state object is conformant with the defined [State Synchronization and Advancement Predicate](#).
- The submitted [Predicate](#) unique identifier matches the [Predicate](#) identifier the [IVSM](#) is based on.
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the target [IVSM](#).
- The digital signature over the state content is valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the [IVSM](#) for that BPI Subject

[R163] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to perform the "Commit State" operation.

Test Steps:

1. Trigger the "Commit State" operation with a state object that conforms to the defined [State Synchronization and Advancement Predicate](#).
2. Trigger the "Commit State" operation with a state object that does not conform to the defined [State Synchronization and Advancement Predicate](#).
3. Check the unique identifier of the submitted [Predicate](#) and the [Predicate](#) identifier that the [IVSM](#) is based on.
4. Look for the unique identifier of the invoking BPI Subject in the list of authorized BPI Subjects on the target [IVSM](#).
5. Confirm that the digital signature over the state content is valid.
6. Confirm that the public key used in the digital signature is cryptographically tied to the unique identifier of the invoking BPI Subject.
7. Check the cryptographically secured and masked secret supplied by the invoking BPI Subject and the one stored in the [IVSM](#) for that BPI Subject.
8. Trigger the "Commit State" operation with a valid state object but an invalid digital signature.
9. Trigger the "Commit State" operation with a valid state object and valid digital signature but with a cryptographically secured and masked secret that does not match the one stored in the [IVSM](#) for that BPI Subject.

Expected Results:

1. The [IVSM](#) accepts the operation.
2. The [IVSM](#) should reject the operation, indicating that the state object does not meet the [Predicate](#) requirements.
3. The unique identifier matches, indicating consistency between the submitted state and the [IVSM](#)'s [Predicate](#).
4. The invoking BPI Subject is authorized, and the [IVSM](#) accepts the operation.
5. The digital signature is valid, indicating the authenticity and integrity of the submitted state.
6. The public key is correctly tied to the BPI Subject's identifier.
7. The secured and masked secret matches, indicating consistency between the supplied and stored secrets.
8. The [IVSM](#) should reject the operation, indicating that the digital signature is not valid.
9. The [IVSM](#) should reject the operation, indicating a mismatch in the secured and masked secret.

[R164]

For BPI Interoperability, a valid "Commit State" operation a BPI invokes MUST

- Update the joint state object in its state storage according to the rules of the [State Synchronization and Advancement Predicate](#)
- Commit the new state of an [IVSM](#) as a valid, succinct, and efficient zero-knowledge proof of correctness of the new state on the CCSM together with its public input and verification key
- Send that cryptographic proof of correctness of the new state on the [IVSM](#) to the invoking BPI Subject

[R164] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to perform the "Commit State" operation.

Test Steps:

1. Trigger the "Commit State" operation with a valid state object.
2. Check the [IVSM](#)'s state storage for the joint state object.

Non-Standards Track Work Product

3. Check the commitment of the new state of the [IVSM](#) on the CCSM.
4. Send the cryptographic proof of correctness of the new state on the [VSM](#) to the invoking BPI.
5. Trigger the "Commit State" operation without updating the joint state object in the [VSM](#)'s state storage.
6. Trigger the "Commit State" operation without committing the new state as a zero-knowledge proof on the CCSM.
7. Trigger the "Commit State" operation without sending the cryptographic proof to the invoking BPI Subject.

Expected Results:

1. The [IVSM](#) accepts the operation.
2. The joint state object is successfully updated.
3. The commitment is a valid, succinct, and efficient zero-knowledge proof of correctness on the CCSM, including its public input and verification key.
4. The invoking BPI Subject receives the cryptographic proof of correctness of the new state on the [VSM](#).
5. The [IVSM](#) should reject the operation, indicating that the joint state object was not updated.
6. The [IVSM](#) should reject the operation, indicating that the proof of correctness was not committed.
7. The [IVSM](#) should reject the operation, indicating that the proof was not sent to the BPI Subject.

[R165]

The valid zero-knowledge proof of correctness of the new joint state MUST be publicly verifiable on the CCSM upon which the [VSM](#) was instantiated.

[R165] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- A new joint state has been committed on the CCSM (Consensus Controlled State Machine) with a valid zero-knowledge proof.

Test Steps:

1. Retrieve the zero-knowledge proof associated with the commitment of the new joint state on the CCSM.
2. Set up a publicly verifiable tool or service capable of validating zero-knowledge proofs.
3. Input the zero-knowledge proof, public input data, and verification keys into the verification tool or service.
4. Trigger the public verification process using the configured tool or service and check the provided result.

Expected Results:

1. The zero-knowledge proof is obtained.
2. The tool or service is ready for use.
3. The tool is configured with the necessary inputs.
4. The tool performs cryptographic computations and verifies that the zero-knowledge proof is valid and publicly verifiable on the CCSM.

Invite Participants to BPI Interoperability Process

Once an [IVSM](#) has been launched, the initiating BPI Subject can invite the initially specified, authorized BPI Subject from other BPIs.

[R166]

A "Invite Participants to BPI Interoperability Process" operation MUST have the following properties:

- A unique message number
- The unique identifier of the inviting BPI Subject
- The [State Synchronization and Advancement Predicate](#) utilized in the [IVSM](#) for which the invitation was issued.
- A URI to accept or reject the invitation
- An object containing all of the target [IVSM](#)'s endpoints as URIs
- A digital signature of the inviting BPI Subject over the content of the invitation

[R166] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- The inviting BPI Subject is registered and authorized to perform the invitation.
- The [IVSM](#) has a [State Synchronization and Advancement Predicate](#) configured.

Test Steps:

1. The inviting BPI Subject initiates the "Invite Participants to BPI Interoperability Process" operation.
2. Check the invitation's message number.
3. Check the invitation for the unique identifier of the inviting BPI Subject.
4. Check the invitation for the [State Synchronization and Advancement Predicate](#) utilized in the [IVSM](#).
5. Check for a URI in the invitation for accepting or rejecting the invitation.
6. Check that the invitation includes an object containing all of the target [IVSM](#)'s endpoints as URIs.
7. Use the public key associated with the inviting BPI subject verify the digital signature over the content of the invitation.

Expected Results:

1. The invitation process is initiated.
2. The invitation contains a unique and non-repeating message number.
3. The identifier of the inviting BPI Subject is accurately included.
4. The correct State [Predicate](#) is specified in the invitation.
5. A valid URI is provided for participants to respond to the invitation.
6. The URIs of the target [IVSM](#)'s endpoints are accurately specified.
7. The digital signature is valid, confirming the authenticity of the invitation.

[R167]

A "Invite Participants to BPI Interoperability Process" operation MUST satisfy the following conditions to be valid:

- The digital signature over the invitation content is valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The submitted [State Synchronization and Advancement Predicate](#) is conformant to [\[R140\]](#) - [\[R142\]](#).

[R167] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to participate in the BPI Interoperability process.

Non-Standards Track Work Product

Test Steps:

1. Trigger the "Invite Participants to BPI Interoperability Process" operation with a valid digital signature.
2. Trigger the "Invite Participants" operation with an invalid digital signature.
3. Trigger the "Invite Participants" operation with a digital signature whose public key is not cryptographically tied to the unique identifier of the invoking BPI Subject.
4. Trigger the "Invite Participants" operation with a [State Synchronization and Advancement Predicate](#) that is conformant to [R140] - [R142].
5. Trigger the "Invite Participants" operation with a [State Synchronization and Advancement Predicate](#) that does not conform to [R140] - [R142].
6. Trigger the "Invite Participants" operation without including a digital signature.
7. Trigger the "Invite Participants" operation without including a [State Synchronization and Advancement Predicate](#).

Expected Results:

1. The [IVSM](#) accepts the operation as valid.
2. The [IVSM](#) rejects the operation, indicating that the digital signature is not valid.
3. The [IVSM](#) rejects the operation, indicating that the digital signature's public key is not tied to the BPI Subject.
4. The [IVSM](#) accepts the operation as valid.
5. The [IVSM](#) rejects the operation, indicating that the submitted [Predicate](#) is not conformant.
6. The [IVSM](#) rejects the operation, indicating that a valid digital signature is required.
7. The [IVSM](#) rejects the operation, indicating that a conformant [Predicate](#) is required.

Accept/Reject Invite

To accept or reject an invitation, a BPI Subject must invoke the endpoint provided by the invitation operation.

[R168]

A "Accept/Reject Invite" operation MUST have the following properties:

- The unique identifier of the target [IVSM](#)
- The unique identifier of the invited BPI Subject
- An accept or reject value
- The digital signature over the content of the operation

[R168] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to participate in the BPI Interoperability process.
- An invitation has been issued, and the BPI Subject has received the invitation.

Test Steps:

1. Trigger the "Accept/Reject Invite" operation with the accept value.
2. Trigger the "Accept/Reject Invite" operation with the reject value.
3. Confirm that the "Accept/Reject Invite" operation includes the unique identifier of the target [IVSM](#).
4. Confirm that the "Accept/Reject Invite" operation includes the unique identifier of the invited BPI Subject.
5. Confirm that the "Accept/Reject Invite" operation includes the accept or reject value.
6. Confirm that the "Accept/Reject Invite" operation includes the digital signature over the content.
7. Trigger the "Accept/Reject Invite" operation without including the unique identifier of the target [IVSM](#).
8. Trigger the "Accept/Reject Invite" operation without including the unique identifier of the invited BPI Subject.
9. Trigger the "Accept/Reject Invite" operation without including the accept or reject value.
10. Trigger the "Accept/Reject Invite" operation without including the digital signature.

Expected Results:

1. The [IVSM](#) processes the operation, and the BPI Subject is accepted to participate in the BPI Interoperability process.
2. The [IVSM](#) processes the operation, and the invitation is rejected. The BPI Subject is not added to the BPI Interoperability process.
3. The identifier is present and correct.
4. The identifier is present and correct.
5. The value is present and correct, indicating whether the invitation is accepted or rejected.
6. The digital signature is present and can be verified with the BPI Subject's public key.
7. The [IVSM](#) rejects the operation, indicating that the identifier is required.
8. The [IVSM](#) rejects the operation, indicating that the identifier is required.
9. The [IVSM](#) rejects the operation, indicating that the value is required.
10. The [IVSM](#) rejects the operation, indicating that a valid digital signature is required.

[R169]

A "Accept/Reject Invite" operation MUST satisfy the following conditions to be valid:

- The digital signature over the invitation content is valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the [IVSM](#)
- The unique [IVSM](#) identifier provided by the invoking BPI Subject matches the unique identifier of the target [IVSM](#)

[R169] Testability:

Preconditions:

- The [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- BPI (Baseline Protocol Implementation) Subjects are registered and authorized to participate in the BPI Interoperability process.
- An invitation has been issued, and the BPI Subject has received the invitation.

Test Steps:

1. Trigger the "Accept/Reject Invite" operation with a valid digital signature.
2. Trigger the "Accept/Reject Invite" operation (reject) with an invalid digital signature.
3. Use a digital signature-verification function in a cryptographic library with the digital signature, the public key, and the invite content as inputs.
4. Confirm that the "Accept/Reject Invite" operation includes a digital signature whose public key is cryptographically tied to the unique identifier of the invoking BPI Subject.
5. Check the content of the invitation for the unique identifier of the invoking BPI subject. Then, look for this identifier in the list of authorized BPI Subjects on the [IVSM](#).
6. Check the [IVSM](#) identifier provided by the invoking BPI Subject and the identifier of the target [IVSM](#).
7. Trigger the "Accept/Reject Invite" operation with an invalid digital signature.
8. Trigger the "Accept/Reject Invite" operation with a BPI Subject identifier that is not in the list of authorized BPI Subjects on the [IVSM](#).
9. Trigger the "Accept/Reject Invite" operation with a unique [IVSM](#) identifier that does not match the unique identifier of the target [IVSM](#).

Expected Results:

Non-Standards Track Work Product

1. The [IVSM](#) processes the operation, and the BPI Subject is accepted to participate in the BPI Interoperability process.
2. The [IVSM](#) processes the operation, and the invitation is rejected. The BPI Subject is not added to the BPI Interoperability process.
3. The digital signature can be verified with the BPI Subject's public key.
4. The public key is present and tied to the BPI Subject's unique identifier.
5. The identifier is present and authorized.
6. The identifiers match, indicating that the acceptance or rejection is for the correct [IVSM](#).
7. The [IVSM](#) rejects the operation, indicating that the digital signature is not valid.
8. The [IVSM](#) rejects the operation, indicating that the BPI Subject is not authorized.
9. The [IVSM](#) rejects the operation, indicating that the identifiers do not match.

[R170]

For BPI Interoperability, a valid "Accept/Reject Invite" operation a BPI invokes MUST return from the [IVSM](#) a cryptographically secured and masked secret for the accepting BPI Subject if the invitation is accepted and no value if the invitation is rejected.

[R170] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- The "Accept/Reject Invite" operation is initiated by the inviting BPI Subject.
- The invitation has been sent, and the accepting BPI Subject is ready to respond.

Test Steps:

1. The accepting BPI Subject invokes the "Accept/Reject Invite" operation with the acceptance flag.
2. Check the return value from the [IVSM](#) after accepting the invitation.
3. The accepting BPI Subject invokes the "Accept/Reject Invite" operation with the rejection flag.
4. Check the return value from the [IVSM](#) after rejecting the invitation.

Expected Results:

1. The invitation is accepted.
2. The [IVSM](#) returns a cryptographically secured and masked secret for the accepting BPI Subject.
3. The invitation is rejected.
4. The [IVSM](#) returns no value.

Add/Remove BPI Subject

Adding to and removing from an [IVSM](#) one or more BPI Subjects is expected to be a typical operation given that, for example, joint state finalization requirements may change during processing requiring BPI Subjects to be added or removed. Given that this is a sensitive business operation concerning, in particular, audits, care has to be taken to ensure proper controls.

[R171]

The "Add BPI Subject" operation a BPI invokes MUST be initiated only by an authorized BPI on the [IVSM](#).

[R171] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. An authorized BPI Subject invokes the "Add BPI Subject" operation on the [IVSM](#).
2. An unauthorized BPI Subject attempts to invoke the "Add BPI Subject" operation on the [IVSM](#).

Expected Results:

1. The operation is initiated successfully.
2. The operation is not initiated, and an authorization error is returned.

[R172]

An authorized BPI Subject on an [IVSM](#) MUST only be able to remove itself through the "Remove BPI Subject" operation.

[R172] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. An authorized BPI Subject invokes the "Remove BPI Subject" operation on the [IVSM](#) to remove itself.
2. An authorized BPI Subject attempts to invoke the "Remove BPI Subject" operation on the [IVSM](#) to remove another BPI Subject.

Expected Results:

1. The operation is initiated, and the BPI Subject is successfully removed.
2. The operation is not initiated, and an error is returned indicating that a BPI Subject can only remove itself.

[R173]

The [IVSM](#) MUST NOT be able to prevent a BPI Subject from removing itself from the [IVSM](#).

[R173] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

Non-Standards Track Work Product

1. An authorized BPI Subject invokes the "Remove BPI Subject" operation on the [IVSM](#) to remove itself.
2. An unauthorized BPI Subject attempts to invoke the "Remove BPI Subject" operation on the [IVSM](#) to remove itself.

Expected Results:

1. The operation is initiated, and the BPI Subject is successfully removed.
2. The operation is initiated, and the unauthorized BPI Subject is successfully removed.

[R174]

The "Add BPI Subject" or "Remove BPI Subject" Operation MUST have the following properties:

- The unique identifier of the target [IVSM](#)
- The unique identifier of the invoking BPI Subject
- The unique identifier of the added or removed BPI Subject
- The cryptographically secured and masked secret supplied by the invoking BPI Subject
- The digital signature of the invoking BPI Subject over the content of the operation

[R174] Testability

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. An authorized BPI Subject invokes the "Add BPI Subject" operation on the [IVSM](#) with the correct unique identifier of the target [IVSM](#), invoking BPI Subject, added BPI Subject, cryptographically secured and masked secret, and digital signature.
Expected Result: The operation is initiated with the required properties and carried out successfully.
2. An authorized BPI Subject invokes the "Add BPI Subject" operation on the [IVSM](#) with one or more of the required properties missing or incorrect.
Expected Result: The operation is initiated without all of the required properties and fails.
3. An authorized BPI Subject invokes the "Remove BPI Subject" operation on the [IVSM](#) with the correct unique identifier of the target [IVSM](#), invoking BPI Subject, added BPI Subject, cryptographically secured and masked secret, and digital signature.
Expected Result: The operation is initiated with the required properties and carried out successfully.
4. An authorized BPI Subject invokes the "Remove BPI Subject" operation on the [IVSM](#) with one or more of the required properties missing or incorrect.
Expected Result: The operation is initiated without all of the required properties and fails.

[R175]

The "Add BPI Subject" or "Remove BPI Subject" Operation MUST satisfy the following conditions to be valid:

- The digital signature over the operation's content is valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the target [IVSM](#)
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the target [IVSM](#) for that BPI Subject
- The unique [IVSM](#) identifier provided by the invoking BPI Subject matches the unique identifier of the target [IVSM](#)

[R175] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Use a digital signature-verification function in a cryptographic library with the digital signature, the public key, and the invite content as inputs.
2. Trigger the "Add BPI Subject" or "Remove BPI Subject" Operation with an invalid digital signature.
3. Examine the validity conditions associated with the "Add BPI Subject" operation.
4. Examine the validity conditions associated with the "Remove BPI Subject" operation.

Expected Results:

1. The function's output shows that the digital signature is valid.
2. The operation fails.
3. The operation satisfies the specified conditions, including valid digital signatures, tied public key, authorization check, matching secured secret, and correct [IVSM](#) identifier.
4. The operation satisfies the specified conditions, including valid digital signatures, tied public key, authorization check, matching secured secret, and correct [IVSM](#) identifier.

[R176]

A newly added BPI Subject MUST be approved by a quorum of authorized BPI Subjects on the [IVSM](#).

[R176] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- A quorum of authorized BPI Subjects needed to add a new BPI Subject is defined in the [State Synchronization and Advancement Predicate](#) of the [IVSM](#).

Test Steps:

1. Initiate the "Add BPI Subject" operation with the BPI Subject that will be added.
2. Check if the new BPI Subject is added to the [IVSM](#) before approval from a quorum.
3. Have a group of authorized BPI Subjects that is not enough to meet the quorum approve the addition of the new BPI Subject.
4. Have a group of authorized BPI Subjects that is enough or more than enough to satisfy the quorum approve the addition of the new BPI Subject.
5. Check the status of the quorum approval of the new BPI subject after it is authorized.

Expected Results:

1. The operation is initiated successfully and a new BPI Subject is being added to the [IVSM](#).
2. The [IVSM](#) does not accept the new BPI subject and waits for approval from the quorum.
3. The [IVSM](#) does not accept the new BPI subject and waits for approval from more authorized BPI subjects to satisfy the quorum.
4. The [IVSM](#) acknowledges the approvals and processes the addition of the new BPI Subject.

Non-Standards Track Work Product

5. The [IVSM](#) confirms that the quorum has approved the addition of the new BPI Subject.

[R177]

The quorum required to add a new BPI Subject to an [IVSM](#) MUST be defined in the [State Synchronization and Advancement Predicate](#) of the [IVSM](#).

[R177] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Check the [State Synchronization and Advancement Predicate](#) of the [IVSM](#) for specification of the required quorum.
2. Add a new BPI Subject to the [IVSM](#) by initiating the "Add BPI Subject" operation.
3. Have less authorized BPI Subjects than the specified quorum approve the new BPI subject.
4. Have the quorum of authorized BPI Subjects approve the addition of the new BPI Subject.

Expected Results:

1. The [Predicate](#) specifies the required quorum for adding a new BPI Subject.
2. The operation is initiated successfully.
3. The [IVSM](#) enforces the quorum requirement and doesn't accept the new BPI subject.
4. The [IVSM](#) acknowledges the approvals and processes the addition of the new BPI Subject, showing that it uses the specified quorum.

[R178]

The "Add BPI Subject" operation approved on the [IVSM](#) MUST add the BPI Subject listed in the operation to the [IVSM](#).

[R178] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- The [State Synchronization and Advancement Predicate](#) of the [IVSM](#) defines the required quorum for adding a new BPI Subject.

Test Steps:

1. Initiate the "Add BPI Subject" operation.
2. Have the defined quorum of authorized BPI Subjects approve the addition of the new BPI Subject.
3. Check the list of authorized BPI Subjects on the [IVSM](#) for the unique identifier of the newly-added BPI subject.

Expected Result:

1. The operation is initiated successfully.
2. The [IVSM](#) acknowledges the approvals and processes the addition of the new BPI Subject.
3. The [IVSM](#) includes the unique identifier of the new BPI subject in its list of authorized BPI subjects.

[R179]

The "Remove BPI Subject" operation MUST remove the BPI Subject listed in the operation to the [IVSM](#).

[R179] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Initiate the "Remove BPI Subject" operation.
2. Check the list of authorized BPI Subjects on the [IVSM](#) for the unique identifier of the recently-removed BPI subject.

Expected Results:

1. The "Remove BPI Subject" operation is initiated successfully.
2. The list of authorized BPI Subjects on the [IVSM](#) does not include the unique identifier of the recently-removed BPI subject.

Verify State

Verification of the joint state on an [IVSM](#) is a critical operation to validate the correctness of any joint state changes before these changes are finalized on an [IVSM](#). From an audit point of view, the verification of joint state changes is critical, especially for regulatory compliance of business-sensitive operations.

[R180]

Any authorized BPI Subject on an [IVSM](#) MUST be able to verify the joint state on said [IVSM](#).

[R180] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- Joint state changes have been made on the [IVSM](#) and are ready for verification.

Test Steps:

1. Check the [IVSM](#)'s list of authorized BPI subjects for the BPI's unique identifier.
2. The authorized BPI Subject triggers the "Verify State" operation on the [IVSM](#).
3. Check the results of the "Verify State" operation on joint state of the [IVSM](#).
4. Check the audit trail generated by the [IVSM](#) for the verification operation.

Expected Results:

1. The BPI Subject is authorized, and their credentials are valid.

Non-Standards Track Work Product

2. The operation is initiated successfully.
3. The [IVSM](#) provides verification results indicating the correctness of the joint state changes.
4. The audit trail accurately reflects the details of the verification operation initiated by the authorized BPI Subject.

[R181]

The proof of correctness of any joint state on an [IVSM](#) MUST be verifiable on said [IVSM](#) by an authorized BPI Subject.

[R181] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- A joint state with a verifier system for the proof of correctness is present on the [IVSM](#).

Test Steps:

1. Check for the unique identifier of the BPI in the [IVSM](#)'s list of authorized BPI subjects.
2. Verify the zero-knowledge proof using the verifier system.
3. Review the audit trail generated by the [IVSM](#) for the verification operation.

Expected Results:

1. The BPI Subject is authorized, and their credentials are valid.
2. The proof of correctness is successfully verified by the verifier system.
3. The audit trail accurately reflects the details of the verification by the verifier system.

[R182]

The proof of correctness of the initial joint state after the "Commit State" operation is completed on an [IVSM](#) MUST be publicly verifiable on the chosen CCSM of said [IVSM](#).

[R182] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- A verifier system for proofs of correctness is present on the chosen CCSM.

Test Steps:

1. Use the verifier system to verify the proof of correctness for the initial joint state on the chosen CCSM.
2. Inspect the results of the public verification operation to ensure that the initial joint state's proof of correctness is successfully verified.
3. Review the audit trail generated by the CCSM for the public verification operation.

Expected Results:

1. The proof of correctness is successfully verified.
2. The CCSM provides verification results indicating the correctness of the proof associated with the initial joint state.
3. The audit trail accurately reflects the details of the verification operation for the final joint state.

[R183]

The proof of correctness of the final joint state after the "Finalize State" operation is completed on an [IVSM](#) MUST be publicly verifiable on the chosen CCSM of said [IVSM](#).

[R183] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- The "Finalize State" operation has been successfully completed on the [IVSM](#).
- A verifier system for proofs of correctness is present on the chosen CCSM.

Test Steps:

1. Use the verifier system to verify the proof of correctness for the final joint state on the chosen CCSM.
2. Inspect the results of the public verification operation to ensure that the final joint state's proof of correctness is successfully verified.
3. Review the audit trail generated by the CCSM for the public verification operation.

Expected Results:

1. The proof of correctness is successfully verified.
2. The CCSM provides verification results indicating the correctness of the proof associated with the final joint state.
3. The audit trail accurately reflects the details of the verification operation for the final joint state.

Initial and final state commitments must be anchored on the CCSM to ensure that the committed or finalized state cannot be utilized in fraudulent transactions on a BPI on the anchoring CCSM. Note, that this last statement holds only for the CCSM utilized by the [IVSM](#). Any other CCSMs and BPIs operating on them are typically not aware of any state commitment by BPIs on other CCSMs.

[R184]

The "Verify State" Operation MUST have the following properties:

- The unique identifier of the target [IVSM](#)
- The unique identifier of the invoking BPI Subject
- The cryptographic proof of correctness of the last joint state
- The cryptographically secured and masked secret supplied by the invoking BPI Subject
- The digital signature of the invoking BPI Subject over the content of the operation

[R184] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- A verifier system for cryptographic proofs of correctness of the last joint state is present on the [IVSM](#).

Test Steps:

Non-Standards Track Work Product

1. Use the unique identifier of the target [IVSM](#), the unique identifier of the invoking BPI Subject, the cryptographic proof of correctness of the last joint state, the cryptographically secured and masked secret supplied by the invoking BPI Subject, and the digital signature of the invoking BPI Subject as properties for the "Verify State" Operation.
2. Initiate the "Verify State" Operation with the properties above.
3. Initiate the "Verify State" Operation with any of these properties missing.

Expected Results:

1. The "Verify State" Operation has all of the correct properties.
2. The "Verify State" Operation is successful.
3. The "Verify State" Operation fails.

[\[R185\]](#)

The "Verify State" Operation MUST satisfy the following conditions to be valid:

- The digital signature over the operation's content is valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the target [IVSM](#)
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the target [IVSM](#) for that BPI Subject
- The cryptographic proof of correctness of the last joint state provided by the invoking BPI Subject matches the cryptographic proof of correctness of the last joint state of the target [IVSM](#)

[\[R185\]](#) Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- A verifier system for cryptographic proofs of correctness is present on the [IVSM](#).
- The [IVSM](#) has a [State Synchronization and Advancement Predicate](#) configured.

Test Steps:

1. Find the unique identifier of the invoking BPI Subject and look for it in the list of authorized BPI Subjects on the target [IVSM](#).
2. Compare the cryptographically secured and masked secret supplied by the invoking BPI Subject to the one stored in the target [IVSM](#) for that BPI Subject.
3. Compare the cryptographic proof of correctness of the last joint state provided by the invoking BPI Subject to the cryptographic proof of correctness of the last joint state of the target [IVSM](#).
4. Initiate the "Verify State" Operation using a valid digital signature, the digital signature's public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, the unique identifier of the invoking BPI Subject, the cryptographically secured and masked secret, and the cryptographic proof of correctness of the last joint state.
5. Initiate the "Verify State" Operation with any of the conditions in the last step missing or incorrect.

Expected Results:

1. The unique identifier of the invoking BPI Subject is included in the list of authorized BPI Subjects on the target [IVSM](#).
2. The cryptographically secured and masked secrets are the same.
3. The cryptographic proofs of correctness are the same.
4. The operation runs successfully.
5. The operation fails.

[\[R186\]](#)

For BPI Interoperability, a valid "Verify State" operation which a BPI invokes MUST return from either the [IVSM](#) or the CCSM a value of true or false.

[\[R186\]](#) Testability:

Preconditions:

- The [IVSM](#) is instantiated and operational.
- BPI Subjects are registered and authorized on the [IVSM](#).

Test Steps:

1. Initiate the "Verify State" operation following requirements [\[R184\]](#) and [\[R185\]](#).
2. Check the output of the "Verify State" operation in the CCSM or the [IVSM](#).

Expected Results:

1. The "Verify State Operation" is successfully initiated.
2. The the operation outputs either true or false.

Verify Lock Commitment

A participating BPI Subject needs to know that the initial state commitments from the participating BPI Subjects are locked and cannot be altered while the BPI Interoperability process is occurring. Therefore, verifying the lock commitment of an initial state commit is crucial.

[\[R187\]](#)

Any authorized BPI subject on an [IVSM](#) MUST be able to verify the lock commitment of the initial state commit of a participating BPI Subject to the joint initial state of the BPI Interoperability process.

[\[R187\]](#) Testability:

Preconditions:

- The [IVSM](#) is instantiated and operational.
- BPI Subjects are registered and authorized on the [IVSM](#).

Test Steps:

1. Attempt to initiate and run the "Verify Lock Commitment State" operation with all of the properties specified in [\[R189\]](#) and following all of the conditions in [\[R190\]](#). Repeat this step on all/several authorized BPI Subjects.

Expected Results:

1. The operation is successfully initiated on all BPI Subjects that were tested.

[\[R188\]](#)

The proof of correctness of the lock commitment after the "Commit State" operation is completed on an [IVSM](#) MUST be publicly verifiable on the chosen CCSM of said [IVSM](#).

[\[R188\]](#) Testability:

Non-Standards Track Work Product

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- The "Commit State" operation has been successfully completed on the [IVSM](#).
- A verifier system for proofs of correctness is present on the chosen CCSM.

Test Steps:

1. Use the verifier system on the CCSM to verify the proof of correctness of the lock commitment of the joint state.
2. Check the results of the verifier system to ensure that the proof was successfully verified.

Expected Results:

1. The proof of correctness is successfully verified.
2. The CCSM provides verification results indicating the correctness of the proof associated with the lock commitment of the joint state.

[R189]

The "Verify Lock Commitment State" Operation MUST have the following properties:

- The unique identifier of the target [IVSM](#)
- The unique identifier of the invoking BPI Subject
- A lock commitment of one of the initial joint state contributions
- The cryptographically secured and masked secret supplied by the invoking BPI Subject
- The digital signature of the invoking BPI Subject over the content of the operation

[R189] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- A verifier system for cryptographic proofs of correctness of the lock commitment of the joint state is present on the [IVSM](#).

Test Steps:

1. Use the unique identifier of the target [IVSM](#), the unique identifier of the invoking BPI Subject, the cryptographic proof of correctness of the last joint state, the cryptographically secured and masked secret supplied by the invoking BPI Subject, and the digital signature of the invoking BPI Subject as properties for the "Verify Lock Commitment State" Operation.
2. Initiate the "Verify Lock Commitment State" Operation with the properties above.
3. Initiate the "Verify Lock Commitment State" Operation with any of these properties missing.

Expected Results:

1. The "Verify Lock Commitment State" Operation has all of the correct properties.
2. The "Verify Lock Commitment State" Operation is successful.
3. The "Verify Lock Commitment State" Operation fails.

[R190]

The "Verify Lock Commitment State" Operation MUST satisfy the following conditions to be valid:

- The digital signature over the operation's content is valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the target [IVSM](#)
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the [IVSM](#) for that BPI Subject
- The lock commitment of an initial joint state contribution provided by the invoking BPI Subject matches the lock commitment of one of the initial joint state contributions of the target [IVSM](#)

[R190] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- A verifier system for cryptographic proofs of correctness is present on the [IVSM](#).
- The [IVSM](#) has a [State Synchronization and Advancement Predicate](#) configured.

Test Steps:

1. Find the unique identifier of the invoking BPI Subject and then look for it in the list of authorized BPI Subjects on the target [IVSM](#).
2. Compare the cryptographically secured and masked secret supplied by the invoking BPI Subject to the one stored in the target [IVSM](#) for that BPI Subject.
3. Compare the cryptographic proof of correctness of the last joint state provided by the invoking BPI Subject to the cryptographic proof of correctness of the last joint state of the target [IVSM](#).
4. Initiate the "Verify State" Operation using a valid digital signature, the digital signature's public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, the unique identifier of the invoking BPI Subject, the cryptographically secured and masked secret, and the cryptographic proof of correctness of the last joint state.
5. Initiate the "Verify State" Operation with any of the conditions in the last step missing or incorrect.

Expected Results:

1. The unique identifier of the invoking BPI Subject is included in the list of authorized BPI Subjects on the target [IVSM](#).
2. The cryptographically secured and masked secrets are the same.
3. The cryptographic proofs of correctness are the same.
4. The operation executes successfully.
5. The operation fails.

[R191]

For the purpose of BPI Interoperability, a valid "Verify Lock Commitment State" operation which a BPI invokes MUST return from the CCSM underlying the target [IVSM](#) a value of true or false.

[R191] Testability:

Preconditions:

- The [IVSM](#) is instantiated and operational.
- BPI Subjects are registered and authorized on the [IVSM](#).

Test Steps:

1. Initiate the "Verify Lock Commitment State" operation following requirements [\[R189\]](#) and [\[R190\]](#).
2. Check the output of the "Verify State" operation in the CCSM and the [IVSM](#).

Expected Results:

Non-Standards Track Work Product

1. The "Verify State Operation" is successfully initiated.
2. The operation outputs from both the IVSM and the CCSM are either true or false.

Update State

Once an [IVSM](#) is instantiated and all required initial states committed, a BPI Subject can update the joint state on an [IVSM](#) through the "Update State" operation to contribute to the BPI Interoperability state and process.

[R192]

Any authorized BPI subject on an [IVSM](#) MUST be able to update a joint state on an [IVSM](#).

[R192] Testability:

Preconditions:

- The [IVSM](#) is instantiated and operational.
- BPI Subjects are registered and authorized on the [IVSM](#).

Test Steps:

1. Attempt to initiate and run the "Update State" operation with all of the properties specified in [\[R193\]](#) and following all of the conditions in [\[R194\]](#). Repeat this step on all/several authorized BPI Subjects.

Expected Results:

1. The operation is successfully initiated on all BPI Subjects that were tested.

[R193]

A BPI Interoperability state object utilized in the "Update State" operation to the target [IVSM](#) MUST have the following properties:

- The unique identifier for the [State Synchronization and Advancement Predicate](#) of the state to be updated
- The cryptographic proof of correctness of the last joint state
- The unique identifier of the BPI Subject within the context of the originating BPI who updates the state
- The cryptographically secured and masked secret of the invoking BPI Subject
- A creation date
- All required private input data to the program updating the joint state object and creating a zero-knowledge proof of correctness of the updated state object
- All public input data to the program updating the joint state object and creating a zero-knowledge proof of correctness of the updated state object
- A digital signature over the state content tied to a public key associated with the BPI Subject committing the state

[R193] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- A verifier system for cryptographic proofs of correctness of the lock commitment of the joint state is present on the [IVSM](#).

Test Steps:

1. Use the unique identifier of the target [IVSM](#), the unique identifier of the invoking BPI Subject, the cryptographic proof of correctness of the last joint state, the cryptographically secured and masked secret supplied by the invoking BPI Subject, the creation date, all required private input data, all public input data, and the digital signature of the invoking BPI Subject as properties for the "Update State" Operation.
2. Initiate the "Update State" Operation with the properties above.
3. Initiate the "Update State" Operation with any of these properties missing.

Expected Results:

1. The "Update State" Operation has all of the correct properties.
2. The "Update State" Operation is successful.
3. The "Update State" Operation fails.

[R194]

An "Update State" operation MUST satisfy the following conditions to be valid:

- The submitted state object is conformant with the target [IVSM](#)'s defined [State Synchronization and Advancement Predicate](#)
- The submitted [Predicate](#) unique identifier matches the [Predicate](#) identifier the target [IVSM](#) is based on
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the target [IVSM](#)
- The cryptographic proof of correctness of the last joint state submitted matches the cryptographic proof of correctness of the last joint state on the target [IVSM](#)
- The digital signature over the state content is valid:
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the target [IVSM](#) for that BPI Subject

[R194] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- A verifier system for cryptographic proofs of correctness is present on the [IVSM](#).
- The [IVSM](#) has a [State Synchronization and Advancement Predicate](#) configured.

Test Steps:

1. Find the unique identifier of the invoking BPI Subject and then look for it in the list of authorized BPI Subjects on the target [IVSM](#).
2. Compare the cryptographically secured and masked secret supplied by the invoking BPI Subject to the one stored in the target [IVSM](#) for that BPI Subject.
3. Compare the cryptographic proof of correctness of the last joint state provided by the invoking BPI Subject to the cryptographic proof of correctness of the last joint state on the target [IVSM](#).
4. Check the submitted state object with the conditions of the [State Synchronization and Advancement Predicate](#).
5. Find the [Predicate](#) unique identifier the [IVSM](#) is based on and compare it to the submitted unique identifier.
6. Initiate the "Update State" Operation using the state object, [Predicate](#) unique identifier, valid digital signature, digital signature's public key that is cryptographically tied to the unique identifier of the invoking BPI Subject, unique identifier of the invoking BPI Subject, and cryptographically secured and masked secret.
7. Initiate the "Update State" Operation with any of the conditions in the last step missing or incorrect.

Expected Results:

1. The unique identifier of the invoking BPI Subject is included in the list of authorized BPI Subjects on the target [IVSM](#).
2. The cryptographically secured and masked secrets are the same.
3. The cryptographic proofs of correctness are the same.

Non-Standards Track Work Product

4. The submitted state object is conformant with the target [IVSM](#)'s defined [State Synchronization and Advancement Predicate](#).
5. The submitted [Predicate](#) unique identifier matches the [Predicate](#) identifier the target [IVSM](#) is based on
6. The operation executes successfully.
7. The operation fails.

[R195]

For BPI Interoperability, a valid "Update State" operation that a BPI invokes MUST update the target [IVSM](#)'s state storage according to the rules of the [State Synchronization and Advancement Predicate](#) and return the updated joint state, and its cryptographic zero-knowledge proof of correctness including all public inputs and required keys for proof verification.

[R195] Testability:

Preconditions:

- The [IVSM](#) is instantiated and operational.
- BPI Subjects are registered and authorized on the [IVSM](#).
- The joint state object and its cryptographic proof are available for updating.

Test Steps:

1. Check the [IVSM](#)'s state storage.
2. Initiate the "Update State" operation so that it is conformant to requirements [\[R193\]](#) and [\[R194\]](#).
3. Check the [IVSM](#)'s state storage.
4. Check the output from the operation.

Expected Results:

1. The state storage has many states stored on it leading up to and including the current state.
2. The "Update State" operation executes successfully.
3. The state storage is updated according to the rules of the [State Synchronization and Advancement Predicate](#) to include the newest state added from the "Update State" operation.
4. The operation returns the updated joint state, its cryptographic proof of correctness, all public inputs, and the required keys for proof verification.

Accept/Reject State Update

An updated joint state can only be finalized or further advanced if a pre-defined quorum of all participating BPI Subjects agrees on this new joint state on an [IVSM](#).

[R196]

An [IVSM](#) after a joint state update operation has been completed MUST generate a notification to all authorized BPI Subjects with an endpoint as a URI to accept/reject a joint state update as well as the complete new joint state object.

[R196] Testability:

Preconditions:

- The [IVSM](#) is instantiated and operational.
- BPI Subjects are registered and authorized on the [IVSM](#).
- The joint state has been successfully updated through the "Update State" operation.

Test Steps:

1. Initiate the "Update State" operation with properties and conditions conformant to requirements [\[R193\]](#) and [\[R194\]](#).
2. Check one of the authorized BPI Subjects on the [IVSM](#) for notifications from the "Update State" operation.
3. Access the endpoint specified by the URI in the notification.

Expected Results:

1. The "Update State" operation runs successfully.
2. The operation sends a notification to all authorized BPI Subjects with a URI.
3. The URI leads to the endpoint where the BPI Subject is able to accept or reject the joint state update as well as the complete new joint state object.

[R197]

A quorum of authorized BPI Subjects MUST accept the new joint state before it is finalized by the requesting [IVSM](#).

[R197] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- A quorum of authorized BPI Subjects needed to add a new BPI Subject is defined in the [State Synchronization and Advancement Predicate](#) of the [IVSM](#).

Test Steps:

1. Initiate the "Update State" operation with properties and conditions conformant to requirements [\[R193\]](#) and [\[R194\]](#).
2. Check if the joint state of the [IVSM](#) was updated before approval from a quorum.
3. Have a group of authorized BPI Subjects that is not enough to meet the quorum approve the joint state update.
4. Have a group of authorized BPI Subjects that is enough or more than enough to satisfy the quorum approve the joint state update.
5. Check the status of the quorum's approval of the joint state update after it is finalized.

Expected Results:

1. The operation is initiated successfully and a joint state update is waiting for approval from the quorum.
2. The [IVSM](#) does not accept the joint state update and waits for approval from the quorum.
3. The [IVSM](#) does not accept the joint state update and waits for approval from more authorized BPI subjects to satisfy the quorum.
4. The [IVSM](#) acknowledges the approvals and finalizes the joint state update.
5. The [IVSM](#) confirms that the quorum has approved the joint state update.

[R198]

The quorum required to accept or reject a joint state update on an [IVSM](#) MUST be defined in the [State Synchronization and Advancement Predicate](#) of said [IVSM](#).

[R198] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.

Non-Standards Track Work Product

- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Check the [State Synchronization and Advancement Predicate](#) of the [IVSM](#) for specification of the quorum required to accept or reject a joint state update.
2. Initiate the "Update State" operation with properties and conditions conformant to requirements [\[R193\]](#) and [\[R194\]](#).
3. Have less authorized BPI Subjects than the specified quorum approve the joint state update.
4. Have the quorum of authorized BPI Subjects approve the joint state update.

Expected Results:

1. The [Predicate](#) specifies the required quorum for accepting or reject a joint state update.
2. The operation is initiated successfully.
3. The [IVSM](#) enforces the quorum requirement and doesn't accept the joint state update.
4. The [IVSM](#) acknowledges the acceptances and processes the joint state update, showing that it uses the specified quorum.

[\[R199\]](#)

The joint state object on an [IVSM](#) MUST be updated on said [IVSM](#) based on each received BPI Subject vote, either accept or reject.

[\[R199\]](#) Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- The joint state object on the [IVSM](#) has a counter for accept and reject votes.

Test Steps:

1. Use an authorized BPI Subject to vote to accept a joint state update and check the vote counter on the joint state object.
2. Use an authorized BPI Subject to vote to reject a joint state update and check the vote counter on the joint state object.

Expected Results:

1. The counter for accept votes on the joint state object is updated and one higher than the previous count.
2. The counter for reject votes on the joint state object is updated and one higher than the previous count.

[\[R200\]](#)

Once the updated joint state is either accepted or rejected based on the defined quorum, an [IVSM](#) MUST:

- Notify all BPI Subjects if the joint state has been finalized based on the rules of the [State Synchronization and Advancement Predicate](#) of the [IVSM](#)
- Cryptographically seal the joint state such that no further updates to the joint state can be processed

[\[R200\]](#) Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).
- The quorum has finished voting on whether a joint state update will be accepted or rejected.

Test Steps:

1. Check BPI Subjects in the [IVSM](#) for a notification about the joint state update.
2. Attempt to run the "Accept/Reject State Update" operation again on the same joint state.

Expected Results:

1. All BPI Subjects checked have received a notification about the finalization of the joint state.
2. The [IVSM](#) blocks further updates to the joint state once it has been finalized.

[\[R201\]](#)

The "Accept/Reject State Update" Operation MUST have the following properties:

- The unique identifier of the target [IVSM](#)
- The unique identifier of the invoking BPI Subject
- An accept or reject value
- The cryptographically secured and masked secret supplied by the invoking BPI Subject
- The digital signature of the invoking BPI Subject over the content of the operation

[\[R201\]](#) Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Use the unique identifier of the target [IVSM](#), the unique identifier of the invoking BPI Subject, an accept or reject value, the cryptographically secured and masked secret supplied by the invoking BPI Subject, and the digital signature of the invoking BPI Subject over the content of the operation as properties for the "Accept/Reject State Update" operation.
2. Initiate the "Accept/Reject State Update" Operation with the properties above.
3. Initiate the "Accept/Reject State Update" Operation with any of these properties missing.

Expected Results:

1. The "Accept/Reject State Update" Operation has all of the correct properties.
2. The "Accept/Reject State Update" Operation is successful.
3. The "Accept/Reject State Update" Operation fails.

[\[R202\]](#)

The "Accept/Reject State Update" Operation MUST satisfy the following conditions to be valid:

- The digital signature over the operation's content is valid

Non-Standards Track Work Product

- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the target [IVSM](#)
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the target [IVSM](#) for that BPI Subject

[R202] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Find the unique identifier of the invoking BPI Subject and then look for it in the list of authorized BPI Subjects on the target [IVSM](#).
2. Compare the cryptographically secured and masked secret supplied by the invoking BPI Subject to the one stored in the target [IVSM](#) for that BPI Subject.
3. Initiate the "Accept/Reject State Update" Operation using a valid digital signature, a public key for the digital signature that is cryptographically tied to the unique identifier of the invoking BPI Subject, the unique identifier of the invoking BPI Subject, and the cryptographically secured and masked secret supplied by the invoking BPI Subject.
4. Initiate the "Accept/Reject State Update" Operation with any of the conditions in the last step missing or incorrect.

Expected Results:

1. The unique identifier of the invoking BPI Subject is included in the list of authorized BPI Subjects on the target [IVSM](#).
2. The cryptographically secured and masked secrets are the same.
3. The operation executes successfully.
4. The operation fails.

[R203]

For the purpose of BPI Interoperability, a valid "Accept/Reject State Update" operation that a BPI invokes MUST return a value of joint state processing success or failure, and if the joint state has been finalized based on the rules of the [State Synchronization and Advancement Predicate](#) of the target [IVSM](#).

[R203] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Invoke the "Update State" operation on the [IVSM](#) with an authorized BPI Subject.
2. Have the quorum of authorized BPI Subjects vote to reject the joint state update.
3. Access the joint state on the [IVSM](#).
4. Invoke the "Update State" operation on the [IVSM](#) with an authorized BPI Subject.
5. Have the quorum of authorized BPI Subjects vote to accept the joint state update.
6. Access the joint state on the [IVSM](#).

Expected Results:

1. The operation executes successfully, and a vote starts to accept or reject the joint state update.
2. The joint state update is rejected.
3. The joint state remains the same and doesn't get finalized.
4. The operation executes successfully, and a vote starts to accept or reject the joint state update.
5. The joint state update is accepted and finalized.
6. The joint state includes the changes from the update, indicating that the joint state and the "Update State" operation are finalized.

Exit BPI Interoperability

The exit operation can be invoked at any time during the lifecycle of an [IVSM](#) if a BPI Subject determines that something is occurring that puts their business at risk, or if the joint state has been finalized according to rules of the [State Synchronization and Advancement Predicate](#) of an [IVSM](#).

[R204]

Each BPI Subject MUST be able to invoke the "Exit BPI Interoperability" operation at any point in time after the target [IVSM](#) has been instantiated.

[R204] Testability:

Preconditions:

- The target [IVSM](#) is instantiated and operational.
- BPI Subjects are registered and authorized on the [IVSM](#).

Test Steps:

1. Invoke the "Exit BPI Interoperability" operation using any BPI Subject in the [IVSM](#).

Expected Results:

1. The operation executes successfully.

[R205]

The "Exit BPI Interoperability" Operation MUST have the following properties:

- The unique identifier of the target [IVSM](#)
- The unique identifier of the invoking BPI Subject
- The cryptographically secured and masked secret supplied by the invoking BPI Subject
- The digital signature of the invoking BPI Subject over the content of the operation

[R205] Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

Non-Standards Track Work Product

1. Use the unique identifier of the target [IVSM](#), the unique identifier of the invoking BPI Subject, the cryptographically secured and masked secret supplied by the invoking BPI Subject, and the digital signature of the invoking BPI Subject over the content of the operation as properties for the "Exit BPI Interoperability" operation.
2. Initiate the "Exit BPI Interoperability" Operation with the properties above.
3. Initiate the "Exit BPI Interoperability" Operation with any of these properties missing.

Expected Results:

1. The "Exit BPI Interoperability" Operation has all of the correct properties.
2. The "Exit BPI Interoperability" Operation is successful.
3. The "Exit BPI Interoperability" Operation fails.

[\[R206\]](#)

The "Exit BPI Interoperability" Operation MUST satisfy the following conditions to be valid:

- The digital signature over the operation's content is valid
- The digital signature's public key is cryptographically tied to the unique identifier of the invoking BPI Subject
- The unique identifier of the invoking BPI Subject is in the list of authorized BPI Subjects on the target [IVSM](#)
- The cryptographically secured and masked secret supplied by the invoking BPI Subject matches the one stored in the target [IVSM](#) for that BPI Subject

[\[R206\]](#) Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Find the unique identifier of the invoking BPI Subject and then look for it in the list of authorized BPI Subjects on the target [IVSM](#).
2. Compare the cryptographically secured and masked secret supplied by the invoking BPI Subject to the one stored in the target [IVSM](#) for that BPI Subject.
3. Initiate the "Exit BPI Interoperability" Operation using a valid digital signature, a public key for the digital signature that is cryptographically tied to the unique identifier of the invoking BPI Subject, the unique identifier of the invoking BPI Subject, and the cryptographically secured and masked secret supplied by the invoking BPI Subject.
4. Initiate the "Exit BPI Interoperability" Operation with any of the conditions in the last step missing or incorrect.

Expected Results:

1. The unique identifier of the invoking BPI Subject is included in the list of authorized BPI Subjects on the target [IVSM](#).
2. The cryptographically secured and masked secrets are the same.
3. The Operation executes successfully.
4. The Operation fails.

[\[R207\]](#)

For BPI Interoperability, a valid "Exit BPI Interoperability" operation that a BPI invokes MUST return

- A value of processing success or failure

When successful:

- The current joint state object
- The proof of correctness of the last joint state
- The public input to the proof
- The verification keys
- The verification program

[\[R207\]](#) Testability:

Preconditions:

- An [IVSM](#) (Interoperability Virtual State Machine) is instantiated and operational.
- Authorized BPI Subjects are registered with the [IVSM](#).

Test Steps:

1. Invoke the "Exit BPI Interoperability" operation on the [IVSM](#) with properties and conditions conformant to [\[R205\]](#) and [\[R206\]](#).
2. Check any output from the operation on the BPI Subject.
3. Invoke the "Exit BPI Interoperability" operation on the [IVSM](#) with properties and conditions that are not conformant to [\[R205\]](#) and [\[R206\]](#).
4. Check any output from the operation on the BPI Subject.

Expected Results:

1. The "Exit BPI Interoperability" operation runs successfully.
2. The operation outputs the current joint state object, the proof of correctness of the last joint state, the public input to the proof, the verification keys, the verification program.
3. The "Exit BPI Interoperability" operation fails.
4. The operation outputs an error message indicating that is failed.

This completes the specification of the Bi- and Multi-directional BPI interoperability operations.

5.6 Standardized Set of BPI Interoperability APIs

The detailed API specification of the Mono-directional and Bi-/Multi-directional BPI Interoperability operations is out-of-scope for the current version of the Baseline Protocol standard, and will be given in a later version of the Baseline Protocol API specification.

[\[R208\]](#)

There MUST be a set of BPI APIs supporting the BPI Interoperability operations enumerated in requirements [\[R136\]](#) and [\[R139\]](#).

[\[R208\]](#) Testability:

See testability statement for [\[R136\]](#) and [\[R139\]](#).

5.7 BPI Interoperability: Discoverable Standard Transport Security

[\[R209\]](#)

Data in transit between BPIs MUST be encrypted.

[R209] Testability:

Test Preconditions:

- BPI communication infrastructure is set up and operational.
- Cryptographic keys for encryption and decryption are available and properly configured.

Test Steps:

1. The test system initiates communication between two BPIs (source and target).
2. During the communication, the data being transmitted is intercepted.
3. The intercepted data is analyzed to determine if it is encrypted.
4. If the intercepted data is encrypted, proceed to step 6. If not, proceed to step 5.
5. The test system logs a failure and concludes that the data in transit is not properly encrypted.
6. The encrypted data is decrypted at the target BPI using the appropriate cryptographic keys.
7. The decrypted data is analyzed to verify its content and integrity.

Passing Criteria:

- The intercepted data is found to be encrypted.
- The decrypted data is successfully verified for content and integrity.

[R210]

BPI Communication for BPI Interoperability services MUST satisfy all requirements in section [5.3 BPI Service Orchestration](#).

[R210] Testability:

See the Testability statements in section [5.3 BPI Service Orchestration](#).

6 Agreement Execution

Agreement execution within the context of this document is the deterministic state transition from state A to state B of a state object in a BPI, and where the state object represents a valid agreement state between agreement counterparties. A valid agreement state represents a data set that has been obtained from the correct application by a BPI of the agreement rules and data to a set of input data, the output of which has been agreed upon by the counterparties to the agreement. Agreement execution occurs in the the BPI Processing Layer as defined in section [2 Design and Architecture](#).

Note that a deterministic state transition in the context of this document is facilitated by a BPI Workstep within a BPI workflow. Also, note that a BPI Workflow is the execution of a series of causally connected and deterministic BPI Worksteps where the agreement counterparties are grouped into one or more BPI Workgroups that are attached to the BPI worksteps of the BPI workflow. A BPI Workstep, a BPI Workflow, and a BPI Workgroup will be referred to as workstep, workflow and workgroup subsequently. See Figure 9 below for a conceptual view of the relationship between workflow, workgroups, and worksteps, and BPI processing.

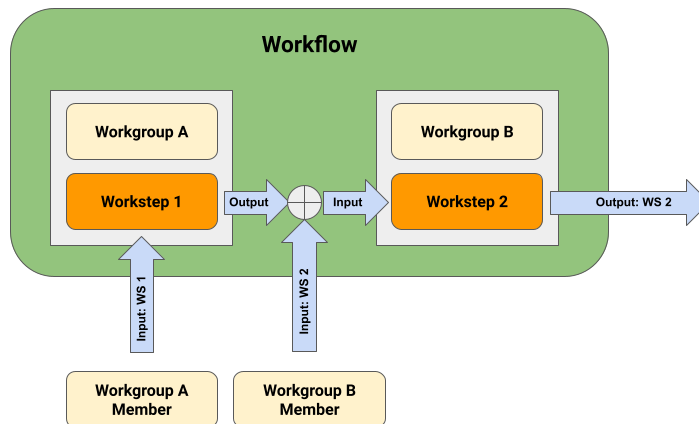


Figure 9: Conceptual View of Workflow, Workstep and Workgroup

6.1 BPI Workstep

First, this document will discuss the requirements for worksteps that will be implemented in the Virtual State Machine of the BPI Processing Layer. Note that strictly speaking one needs to differentiate between the workstep as a logical construct, and its instantiation within a BPI which is called a workstep instance. In the following, and unless required for disambiguation, this document shall use workstep also to mean workstep instance.

[R211]

A workstep MUST have an input, one or more process steps, and an output.

This is just a well-known convention from business process management frameworks.

[R211] Testability: A valid workstep is a function that transforms a given input, using one or more transformations of the input into an output. An invalid workstep meets any of the following conditions:

- it does not have an input, or
- it does not transform the input, or
- it does not have an output.

[R212]

The input of a workstep MUST represent a new, proposed state of a state object compliant with the agreement between the agreement counterparties.

Non-Standards Track Work Product

[R212] Testability: The input and output of a workstep can be formatted according to the data schema of a state object that represents the state of a given agreement.

If the input and output are valid representations of a state according to the agreement, and are not equal, they meet the requirement.

If they are equal, or cannot be represented as a valid state object, they do not meet the requirement.

[R213]

The process steps in a workstep MUST represent a verification system comprised of the set, or subset, of agreement rules and agreement data such that an input can be validated to comply with the agreement rules and agreement data, or not.

[R213] Testability: A verification system is comprised of one or more individual rules as a sequence of evaluation steps, 1 through N. Each evaluation step can be evaluated to a boolean (true or false) using a comparison operator of the step input against a known state or fact such as input value = 1 for evaluation step 1. The final evaluation of the verification system is the product of all the boolean outputs of each evaluation step. The product will, therefore, evaluate to another boolean (true, if all evaluations steps yield a true output, or false, if one evaluation step output is false). Since an agreement rule can be expressed as an evaluation step, so can the set, or subset, of agreement rules and agreement data. Therefore, process steps based on agreement rules that evaluate to either true or false represent a valid verification system. If process steps based on agreement rules that do not evaluate to true or false are not a valid verification system.

[R214]

The output of a workstep MUST represent the verifiable validation result of an input into a workstep as a correct or incorrect new agreement state.

Note that a new agreement state after a correct workstep execution is defined as

New Agreement State = Old Agreement State + Agreed upon New State Object + Workstep Output

See sections 6.4 and 6.5 below on BPI Accounts associated with state objects and BPI transactions for details.

[R214] Testability: IF an input is provided to a workstep, THEN the output of the workstep MUST represent the validation result of the input as a correct or incorrect new agreement state. AND the validation result MUST be verifiable. To verify the validation result, the output of the workstep should contain information that allows the result to be tested or confirmed using a predefined method or standard. This may involve providing a reference to the criteria or rules used for validation, or specifying the expected behavior of the new agreement state.

[R215]

A workstep instance MUST be associated with only one workgroup.

[R215] Testability: A test of this requirement can be comprised of the following sequence:

1. Create multiple work step instances and assign them to different workgroups.
2. Verify that each work step instance is associated with only one workgroup through a system search that only yields one result for each workstep instance.
3. Attempt to assign a work step instance to multiple workgroups.
4. Verify that the system prevents the work step instance from being associated with more than one workgroup by creating an error warning and aborting the system process.
5. Attempt to create a work step instance without specifying a workgroup.
6. Verify that the system prompts the user to select a workgroup before proceeding with the creation of the work step instance.
7. Verify that the system prevents the creation of a work step instance without associating it with a workgroup by creating an error warning and not creating the work step instance.
8. Attempt to associate a work step instance with a non-existent workgroup.
9. Verify that the system prompts the user to select a valid workgroup before proceeding with the association of the work step instance by creating an error warning and not proceeding with the system process..
10. Verify that the system prevents the association of a work step instance with a non-existent workgroup by creating an error warning and aborting the system process.

If all of the above test steps are successfully completed, then the requirement is met

[R216]

A workstep instance MUST inherit the security and privacy policies of its associated workgroup.

See the details of workgroups and their security and privacy policies in [Section 6.3](#).

[R216] Testability: Policies such as an authentication requirement of a BPI Subject are expressible as functions evaluating to true or false, and, therefore, represent worksteps that can be applied to input data in a workstep instance. Furthermore, [R215] expresses the testable association of one or more workstep instances and a workgroup. Therefore, one or more workgroup policies can be evaluated as workstep instances in the context of any workstep instance associated with a workgroup before that workstep instance is executed. Hence, the execution of a given workstep instance associated with a workgroup is contingent upon the output of the evaluation of one or more workgroup policies as workstep instances – the workstep instance is executed if the individual policies together evaluate to true, or is not executed if the policies together evaluate to false either result meets the requirement. If a set of workgroup policies is not successfully associated with a workstep instance associated with the workgroup with those policies, then the workstep instance is executed independent of the evaluation result of the workgroup policies' workstep instances which does not meet the requirement.

[R217]

A workstep MUST have a unique identifier within a BPI.

[R217] Testability:

Preconditions:

- A BPI test instance is set up and operational.
- There are at least two worksteps set up within the BPI.

Test steps:

1. Retrieve the list of worksteps within the BPI.
2. For each workstep in the list, validate that the workstep identifiers are not duplicates.
3. Create a new workstep within the BPI and verify that its identifier does not exist in the list of other workstep identifiers.
4. Attempt to create a new workstep within the BPI with the same identifier as an existing workstep and verify that it is rejected.

Test Passing criteria:

- Test step 2 passes if all worksteps in the list have a unique identifier within the BPI.
- Test step 3 passes if the new workstep is assigned a unique identifier.
- Test step 4 passes if the attempt to create a workstep with the same identifier as an existing workstep is rejected.

[R218]

A workstep MUST be updatable.

[R218] Testability:

Preconditions:

- A BPI test instance is set up and operational.

Non-Standards Track Work Product

- A workstep exists in the BPI system.

Test Steps:

1. Access the workstep in the BPI system.
2. Attempt to update the workstep with new information.
3. Save the changes to the workstep.
4. Access the workstep in the BPI system again.
5. Verify that the new workstep information is displayed correctly

Test Passing Criteria:

- The changes to the workstep are saved successfully and can be viewed when accessing the workstep again.

[R219]

A workstep instance MUST NOT be updated while the workstep instance is being executed by the BPI.

This ensures that no breaking changes with potentially significant negative business impact are introduced while a workstep instance is being executed.

[R219] Testability:

Preconditions:

- A BPI test instance is set up and operational.
- A BPI with at least one workstep defined.
- A workstep instance that is currently being executed by the BPI.

Test steps:

1. Get the identifier of the workstep instance that is currently being executed by the BPI.
2. Attempt to update the workstep instance while it is being executed by the BPI.
3. Verify that the workstep instance was not updated and that an appropriate error message is returned.
4. Wait for the workstep instance to complete execution.
5. Verify that the workstep instance can now be updated.

Test Passing criteria:

- Step 3 must return an appropriate error message indicating that the workstep instance cannot be updated while it is being executed by the BPI.
- Step 5 must successfully update the workstep instance.

[R220]

A workstep MUST be versioned within a BPI.

Note that versions of the same workstep do not have to be compatible with one another.

[R220] Testability:

Preconditions:

- A BPI test instance is set up and operational.
- A BPI has been created and contains at least one workstep.
- The BPI allows for versioning of worksteps.

Test Steps:

1. Create a new workstep within the BPI.
2. Verify that the workstep has been assigned a version identifier (e.g., 1.0 or VABC100).
3. Make a functional change to the workstep and save the changes.
4. Verify that the workstep's version identifier has been updated (e.g., 1.1 or VABC101).
5. Repeat steps 3-4 for at least two more functional changes to the workstep.
6. Verify that each functional change results in an updated version identifier.
7. Attempt to execute the workstep with an earlier version identifier (e.g., 1.0 or VABC100).
8. Verify that an error or warning is thrown indicating that the workstep version is outdated and needs to be updated.
9. Create a new version of the workstep.
10. Verify that the new version has a unique version identifier.
11. Attempt to execute the workstep with the new version identifier.
12. Verify that the workstep executes successfully.

Test Passing Criteria:

- The workstep versioning functionality allows for changes to be made to a workstep while maintaining a record of the changes and ensuring that an outdated version cannot be executed.
- The test passes if all steps execute successfully and the workstep can be versioned and executed without errors or warnings.

[R221]

A workstep MUST be executed inside a BPI's Virtual State Machine (VSM).

Note, that a BPI's Virtual State Machine is part of the BPI Processing Layer as discussed in [Section 2.8.4](#).

[R221] Testability:

Preconditions:

- A BPI test instance is up and running, with the VSM module installed and operational.
- The BPI has at least one workstep defined and versioned.
- The BPI Middleware Layer is functioning correctly and passing inputs and outputs to the VSM.

Test Steps:

1. Select a workstep to execute within the BPI's VSM.
2. Verify that the workstep is versioned by checking the version identifier in the BPI's metadata.
3. Verify that the input to the workstep is being passed correctly from the BPI Middleware Layer to the BPI Processing Layer.
4. Execute the workstep inside the BPI's VSM.
5. Verify that the output generated by the workstep is being passed correctly to the BPI Middleware Layer from the BPI Processing Layer.
6. Repeat steps 1-5 with a different workstep, and verify that each workstep is executed within the BPI's VSM.

Non-Standards Track Work Product

Test Passing Criteria:

- The workstep version identifier is correctly identified and validated.
- The input to the workstep is correctly passed from the BPI Middleware Layer to the VSM.
- The workstep is successfully executed inside the VSM.
- The output generated by the workstep is correctly passed from the VSM to the BPI Middleware Layer.
- All worksteps are successfully executed within the BPI's VSM.

[R222]

The input of a workstep instance MUST be submitted by an authorized member of the workgroup attached to that workstep instance.

Note, that this allows for delegation of authorization from the authorization bearing Entity A to Entity B, akin to a power-of-attorney. This concept is also known as attenuated authorization.

[R222] Testability:

Preconditions:

- A BPI test instance is set up and running.
- A workgroup and its members are set up in the BPI.
- The workgroup is attached to a workstep instance in the BPI.
- The workgroup's security policy defining workgroup member authorizations is defined in the BPI system.

Test Steps:

1. An unauthorized member of the workgroup tries to submit input to the workstep instance. 2. Verify that the BPI system rejects the input submission and displays an error message. 3. An authorized member of the workgroup submits the input of the workstep instance. 4. Verify that the BPI system accepts the input submission and executes the workstep instance.

Test Passing Criteria:

- Test step 2 passes if the BPI system rejects the input submission and displays an error message.
- Test step 4 passes if the BPI system accepts the input submission and executes the workstep instance.
- The test passes if all the test steps pass.

[R223]

A BPI workstep MUST be deterministic.

This means that for a given input, there can be only one valid output from the workstep generated by the BPI.

[R223] Testability:

Preconditions:

- A BPI test instance is set up and running.
- A BPI workstep has a defined input and output.
- The workstep has been implemented and deployed in the BPI system.
- The input to the workstep is well-defined.

Test Steps:

1. Submit the same input to the workstep instance twice.
2. Record the output generated by the workstep for both input submissions.
3. Compare the output generated by the workstep for both input submissions.
4. If the outputs are identical, the test passes. Otherwise, it fails.

Test Passing Criteria:

- The test passes if the output generated by the workstep is identical for the same input submitted to the workstep instance twice.
- The test fails if the output generated by the workstep is not identical for the same input submitted to the workstep instance twice.

[R224]

The output from a workstep execution MUST be finalized through an agreed-upon quorum of cryptographic signatures of the workgroup participants associated with the workstep.

This means that the output of a workstep execution must be verified and agreed upon by a previously defined number of the workgroup participants.

[R224] Testability:

Preconditions:

- A BPI test instance is in place, and there is at least one workstep defined.
- The workgroup is associated with the workstep and its participants have the necessary access rights and cryptographic tools to create digital signatures.
- An agreed-upon quorum threshold for the required number of cryptographic signatures has been defined as a policy in the workgroup.

Test Steps:

1. Submit input data to initiate the execution of the workstep instance.
2. Execute the workstep, and obtain the output.
3. Request the digital signature from each workgroup participant associated with the workstep.
4. Verify the received signatures using the cryptographic tools agreed upon by the workgroup.
5. Count the number of valid signatures obtained and compare it to the quorum threshold.
6. If the number of valid signatures obtained is equal to or greater than the quorum threshold, mark the test as passed. Otherwise, mark it as failed.

Test Passing Criteria: The test will pass if and only if the following conditions are met,

- The workstep instance was executed successfully.
- The digital signatures obtained from the workgroup participants associated with the workstep are valid and verified using the cryptographic tools agreed upon by the workgroup.
- The number of valid signatures obtained is equal to or greater than the quorum threshold.

[R225]

The output from a workstep execution MUST be a valid zero-knowledge proof of correctness of the new agreement state generated by the BPI executing the workstep (privacy preservation).

A Zero-Knowledge Proof is defined as having to satisfy the following three properties:

- **Completeness:** if the statement is true, an honest verifier, i.e., an entity following the protocol properly, will be convinced of this fact by an honest prover.
- **Soundness:** if the statement is false, no cheating prover can convince an honest verifier that it is true, except with some small probability.

Non-Standards Track Work Product

- **Zero-Knowledge:** *if the statement is true, no verifier learns anything other than the fact that the statement is true. In other words, just knowing the statement (not the secret) is sufficient to construct a scenario that shows that the prover knows the secret. This is formalized by showing that every verifier has some simulator that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the verifier.*

[R225] Testability:

Preconditions:

- A BPI test instance with at least one workstep defined.
- A workgroup associated with the workstep.
- Cryptographic signature mechanism is implemented in the BPI.
- Zero-knowledge proof generation is implemented in the BPI.

Test Steps:

1. Submit input data for a workstep instance by an authorized member of the associated workgroup.
2. Execute the workstep with the submitted input data.
3. Generate a zero-knowledge proof of correctness of the new agreement state.
4. Request the cryptographic signatures from the agreed-upon quorum of workgroup participants.
5. Validate that the received cryptographic signatures are valid and match the workgroup participants associated with the workstep.
6. Validate the zero-knowledge proof of correctness of the new agreement state.
7. Verify that the output generated by the workstep execution matches the expected agreement state.

Test Passing Criteria:

- The workstep output matches the agreement state.
- The cryptographic signatures are valid and match the workgroup participants.
- The zero-knowledge proof of correctness of the new agreement state is valid.

[R226]

A zero-knowledge proof of correctness of a workstep output MUST be non-interactive.

Non-interactive in this context means that there is no interaction between the prover (generating the proof) and the verifier.

[R226] Testability:

Preconditions:

- A BPI test instance with at least one workstep defined.
- The BPI workstep has been designed and implemented with an output that requires a zero-knowledge proof of correctness.
- A zero-knowledge proof system that supports non-interactive proofs has been selected and integrated into the BPI.

Test Steps:

1. Submit a valid input to the workstep.
2. Execute the workstep.
3. Generate a zero-knowledge proof of correctness for the output of the workstep using the selected zero-knowledge proof system.
4. Verify that the proof is valid and shows the correctness of the output generated by the workstep based only on the proof data provided to the verification function from the workstep output.

Test Passing Criteria:

- The workstep output is correctly generated.
- The zero-knowledge proof of correctness is valid and shows the correctness of the output generated by the workstep based only on the proof data provided to the verification function from the workstep output.

[R227]

An input that does not represent a new, valid agreement state of a state object MUST NOT generate a valid zero-knowledge proof of correctness of the input.

[R227] Testability:

Preconditions:

- A BPI test instance is set up and running with at least one workstep that operates on a state object.
- The BPI is configured to generate zero-knowledge proofs of correctness for each workstep output.
- The BPI is configured to reject inputs that do not represent a valid new agreement state.

Test Steps:

1. Submit an input to the workstep that does not represent a valid new agreement state for the state object.
2. Execute the workstep.
3. Verify that the BPI rejects the input and does not generate a valid zero-knowledge proof of correctness for the output.
4. Submit a new input to the workstep that represents a valid new agreement state for the state object.
5. Execute the workstep.
6. Verify that the BPI accepts the input, generates a valid zero-knowledge proof of correctness for the output, and updates the state object accordingly.

Test Passing Criteria:

- The BPI should reject the invalid input and not generate a valid zero-knowledge proof of correctness for the output in step 3.
- The BPI should accept the valid input, generate a valid zero-knowledge proof of correctness for the output, and update the state object accordingly in step 6.

[R228]

A zero-knowledge proof of correctness of a workstep output MUST be verifiable by any 3rd party in a time at most proportional to the size of the prover system that generated the proof.

The time requirement means that any 3rd party verifier must be able to verify the proof representing a prover system of size n in time $O(n)$, e.g., a Merkle-proof of a Merkle-trie branch of 10 tuples can be verified in 10 computational steps. It also means that the zero-knowledge proof of correctness of input does not have to be succinct. Succinct means that the proofs are short (smaller than the size of the prover circuit) and that the verification is fast.

[R228] Testability:

Preconditions:

- A BPI test instance is set up and running with at least one workstep.
- A BPI workstep with an output that can be used to generate a zero-knowledge proof.
- A prover system that can generate a zero-knowledge proof of correctness for the output of size N .
- The size N is given by the size of the circuit used to generate the proof in terms of the number of circuit constraint equations.

Non-Standards Track Work Product

- A verifier system that can verify the zero-knowledge proof.

Test Steps:

1. Submit a valid input to the BPI workstep.
2. Generate a zero-knowledge proof of correctness for the output using the prover system.
3. Verify the zero-knowledge proof using the verifier system.
4. Measure the number of computational steps taken by the verifier system to verify the proof.
5. Verify that the number of computational steps taken to verify the proof is proportional to or less than the size of the prover system.

Test Passing Criteria:

- The test will pass if the zero-knowledge proof of correctness for the workstep output can be verified by the verifier system in a number of computational step at most proportional to the size of the prover system that generated the proof.

[D26]

The zero-knowledge proof of correctness of a workstep output SHOULD be succinct.

A zero-knowledge proof is said to be succinct if the proof is smaller than the size of the circuit, the poly-logarithm to be precise, used to generate the proof.

[D26] Testability:

Preconditions:

- A BPI test instance is set up and running with at least one workstep.
- A BPI workstep with an output that can be used to generate a zero-knowledge proof.
- A prover system that can generate a zero-knowledge proof of correctness for the output of size N.
- The size N is given by the size of the circuit used to generate the proof in terms of the number of circuit constraint equations.
- A verifier system that can verify the zero-knowledge proof.

Test Steps:

1. Submit a valid input to the BPI workstep.
2. Generate a zero-knowledge proof of correctness for the output using the prover system.
3. Verify the zero-knowledge proof using the verifier system.
4. Measure the number of computational steps taken by the verifier system to verify the proof.
5. Verify that the number of computational steps taken to verify the proof is proportional at most to or less than the poly-logarithm of the size of the prover system.

Test Passing Criteria:

- The test will pass if the zero-knowledge proof of correctness for the workstep output can be verified by the verifier system in a number of computational steps at most proportional to the size of the prover system that generated the proof.

[D27]

The zero-knowledge proof of correctness of a workstep output SHOULD be efficient.

Efficient in this context means that the size of the proof does not grow with the size of the prover system(s). This is a highly desirable feature when it comes to both data on a CCSM and verification time.

[D27] Testability:

Preconditions:

- A BPI test instance is set up and running with at least two worksteps.
- BPI worksteps with a valid input, business logic, and output.
- The BPI workstep inputs are the same.
- The business logic in the worksteps is different with the complexity of the business logic increasing with each workstep which leads to different sizes of zero-knowledge proof circuits.
- A zero-knowledge proof system in place for verifying the correctness of the output.

Test steps:

1. Create a workstep instance with a small prover system size and a valid input.
2. Execute the workstep and generate a zero-knowledge proof of correctness of the output.
3. Verify the zero-knowledge proof's correctness using the proof system and record the verification time.
4. Repeat steps 1-3 with workstep instances having progressively larger prover system sizes for each new workstep.
5. Compare the verification times for each workstep instance to determine whether the verification time grows as the size of the prover system grows.

Test Passing criteria:

- The verification time of the zero-knowledge proof should remain constant as the size of the prover system grows, demonstrating that the zero-knowledge proof is efficient.
- If the verification time grows significantly as the size of the prover system grows, the test fails, indicating that the zero-knowledge proof system is not efficient.

[D28]

The zero-knowledge proof of correctness of a workstep output SHOULD be based on modular constructions.

Modular in this context means that the proof system can represent multiple statements, in other words, multiple proofs together, in one proof. For example, "I have an invoice with a face value of over \$10,000, payable within 30 days, and the payee has never missed a payment in 10 years of doing business with me". This is also highly desirable, especially when having to combine various proofs as in the previous statement.

[D28] Testability:

Preconditions:

- A BPI test instance is set up and running with at least two worksteps.
- BPI worksteps with a valid input, business logic, and output.
- A BPI's workstep output must be generated by a zero-knowledge proof generator that claims to use modular constructions.

Test Steps:

1. Create a simple circuit that takes a fixed input and produces a fixed output.
2. Generate two different workstep outputs using the same simple circuit as the business logic.
3. Verify the zero-knowledge proof of correctness for each workstep output separately.
4. Generate a third workstep output that is the result of chaining the two simple circuits together in a new workstep.
5. Verify the zero-knowledge proof of correctness for the third workstep output, which represents the result of chaining two circuits.
6. Compare the verification time for the third workstep output with the verification time for each of the two original workstep outputs.

Non-Standards Track Work Product

Test Passing Criteria: The test passes if,

- The verification time for the third workstep output is not significantly greater than the verification time for each of the two original workstep outputs.
- The verification of the third workstep output is successful and it represents the correct output of the chained circuits.

[R229]

A zero-knowledge proof of correctness of a workstep output that has been finalized on the BPI MUST be committed to the CCSM utilized by the BPI using a compact cryptographic proof.

Such a commitment can represent more than one zero-knowledge proof of correctness of an input. Compact in this context means that the CCSM commitment is smaller in size than the totality of the proof(s) represented by the commitment. This is desirable because it reduces the data footprint of the BPI which should be one of the implementation goals of a BPI. Note that a compact proof may be succinct or efficient.

[R229] Testability:

Preconditions:

- A BPI test instance is set up and running with at least one workstep.
- A BPI workstep with a valid input, business logic, and output.
- The BPI worksteps have been executed and their output has been finalized through an agreed-upon quorum of cryptographic signatures of the workgroup participants associated with the worksteps.

Test Steps:

1. Retrieve the zero-knowledge proof(s) of correctness for the output of the executed workstep.
2. Generate a compact cryptographic proof of the zero-knowledge proof(s) of correctness.
3. Verify that the compact cryptographic proof is smaller in size than the total size of the zero-knowledge proof(s) of correctness.
4. Commit the compact cryptographic proof to the CCSM utilized by the BPI for storage.
5. Verify that the commitment to the proof(s) has been stored on the CCSM.

Test Passing Criteria:

- The compact cryptographic proof is smaller in size than the total size of the zero-knowledge proof(s) of correctness.
- The commitment to the proof(s) has been stored on the CCSM.

[R230]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output on the CCSM MUST be verifiable by any 3rd party at any time in a time at most proportional to the size of the prover system.

Note, that the requirement does not state that the proof has to be verifiable on the CCSM itself, that it does not need to be succinct, and that it does not need to be efficient.

[R230] Testability:

Preconditions:

- A BPI test instance is set up and running with at least one workstep.
- A BPI workstep with a valid input, business logic, and output.
- A BPI with a workstep that generates a cryptographic proof of correctness for a state transition has been executed and its output has been committed to the CCSM.
- The size of the prover system has been determined based on the number of constraint equations generated by the circuit used in the workstep.

Test Steps:

1. Obtain the cryptographic proof of correctness for the state transition from the CCSM.
2. Attempt to verify the proof using a third-party verification tool.
3. Measure the time taken for verification in the number of computational steps to verify the constraint equations.
4. Calculate the proportionality constant between the size of the prover system and the time taken for verification.
5. Verify that the time taken for verification is of order one, $O(1)$.

Test Passing Criteria:

- The test will pass if the verification of the cryptographic proof of correctness of the BPI state transition can be completed by a third party within a time at most proportional to the size of the prover system, as specified by the requirement.

[D29]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output generated by a BPI and stored on the CCSM utilized by the BPI SHOULD be verifiable by any 3rd party and at any time by the CCSM utilized by the BPI.

[D29] Testability:

Preconditions:

- A BPI test instance is set up and running with at least one workstep.
- A BPI workstep with a valid input, business logic, and output.
- A BPI workstep has been executed and the output, along with its cryptographic proof, has been stored on the CCSM utilized by the BPI.
- The cryptographic proof of correctness generated satisfies [R230].

Test Steps:

1. Retrieve the cryptographic proof of the workstep output stored on the CCSM utilized by the BPI.
2. Request the CCSM utilized by the BPI to verify the cryptographic proof of the workstep output.
 3. Verify that the CCSM is able to successfully verify the cryptographic proof.
3. Repeat steps 1-3 for multiple workstep outputs stored on the CCSM utilized by the BPI at different randomly selected points in time during the testing period.

Test Passing Criteria: The test is considered passing if all of the following criteria are met,

- The CCSM utilized by the BPI is able to verify the cryptographic proof of correctness of the BPI state transition.
- The test has been repeated for multiple workstep outputs stored on the CCSM utilized by the BPI, at random points in time during the testing period, and all verifications are successful.

[D30]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output and stored on the CCSM SHOULD be succinct.

[D30] Testability: The same test as for [D26] can be used for [D30] if the cryptographic proof of correctness of a BPI state transition that is stored on the CCSM is a zero-knowledge proof with the same characteristics as in used in [D26].

[D31]

Non-Standards Track Work Product

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output on the CCSM SHOULD be efficient.

[D31] Testability: The same test as for [D27] can be used for [D31] if the cryptographic proof of correctness of a BPI state transition that is stored on the CCSM is a zero-knowledge proof with the same characteristics as in used in [D27].

[D32]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output on the CCSM SHOULD be based on modular constructions.

[D32] Testability: The same test as for [D28] can be used for [D32] if the cryptographic proof of correctness of a BPI state transition that is stored on the CCSM is a zero-knowledge proof with the same characteristics as in used in [D28].

In specific situations, the above SHOULD requirements become MUST.

[O3]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output MAY be used in a commercial value-creation or value-exchange event.

This means that the output of a workstep can be used to represent a unit of value accounting such as a digital asset. For example, one or more workstep outputs may be used as (verifiable) collateral in issuing an asset-backed security (value-creation).

[O3] Testability:

Preconditions:

- A BPI test instance is set up and running with at least one workstep.
- The BPI workstep must have a valid state object associated to it.
- The workstep execution has been performed and the corresponding cryptographic proof of correctness of the state transition has been generated and stored in the CCSM utilized by the BPI.

Test Steps:

1. Initiate a commercial value-creation or value-exchange event using the BPI state object that is a result of the workstep execution for which the cryptographic proof was generated.
2. Verify that the cryptographic proof is used to validate the correctness of the state transition during the event.
3. Verify that the cryptographic proof is available for any 3rd party to verify the correctness of the state transition.

Test Passing Criteria:

- The commercial value-creation or value-exchange event is successfully initiated using the BPI state object.
- The cryptographic proof of correctness of the state transition is used to validate the correctness of the state transition during the event.
- The cryptographic proof is available for any 3rd party to verify the correctness of the state transition.

[CR16]<[O3]

A cryptographic proof of correctness of a BPI state transition used in a commercial value-creation MUST be succinct.

[CR16]<[O3] Testability: The same test as in [D26] Testability may be used.

[CR17]<[O3]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output MUST be individually available on the CCSM utilized by the BPI after it has been finalized on the BPI (Liveness).

The zero-knowledge proof of correctness can be a cryptographic aggregator of proofs of workstep input correctness that would allow multiple proofs to be represented and provable within one proof.

[CR17]<[O3] Testability: The same test as in [R229] Testability may be used with the only difference being that the proof on the CCSM contains the representation of only one proof.

[CR18]<[O3]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output MUST be verifiable by any 3rd party on the CCSM utilized by the BPI (censorship-resistant individual proof verifiability).

[CR18]<[O3] Testability: The same test as in [R230] Testability may be used.

[CR19]<[O3]

A cryptographic proof of correctness of a BPI state transition as expressed by a workstep output MUST NOT be able to be used in more than one commercial value-creation/exchange event at any time.

This requirement is necessary to avoid the usage of the same output as collateral in more than one commercial value-creation event, such as tokenization, and subsequent usage in value-exchange events, also known as double-pledging.

A workstep output according to [CR19]<[O3] will have to be included in a commitment that the output is pledged in a commercial value-creation event. This will be called an "Output Pledge".

Note that this only restricts the usage of pledged outputs to the CCSM utilized in a given BPI. It is impossible to foresee which CCSMs will be used for the implementation of a BPI. Therefore, for example, a document may be baselined on more than one BPI and CCSM without being able to avoid a double pledge since the individual CCSMs are not synchronized.

[CR19]<[O3] Testability:

Preconditions:

- A BPI with the capability of generating cryptographic proofs of correctness of a state transition as expressed by a workstep output.
- An updateable commitment system on the CCSM utilized by the BPI that allows for creating and verifying commitments.
- A commercial value-creation/exchange event where a BPI workstep output may be pledged.
- The BPI workstep output that will be pledged in the commercial value-creation event.

Test Steps:

1. Generate a cryptographic proof of correctness of a BPI state transition as expressed by a workstep output.
2. Initiate a commercial value-creation or value-exchange event using the BPI state object that is a result of the workstep execution for which the cryptographic proof was generated.
3. Verify that the cryptographic proof is used to validate the correctness of the state transition during the event.
4. Verify that the cryptographic proof is available for any 3rd party to verify the correctness of the state transition.
5. Create an output pledge by including the BPI workstep output and a resolvable identifier of the commercial value-creation in a commitment stating that the output is pledged in a commercial value-creation event.
6. Verify that the output pledge is stored on the CCSM utilized by the BPI.
7. Attempt to use the output pledge in another commercial value-creation/exchange event.
8. Verify that the attempt to use the output pledge in another commercial value-creation/exchange event fails because the commercial value-creation event verification that the cryptographic proof of correctness of a BPI state transition included in the value-creation event is not included in the Output Pledge on the BPI's CCSM fails.

Non-Standards Track Work Product

9. Verify that the output pledge is used in the commercial value-creation/exchange event in which it was committed.
10. Verify that the output pledge can be revoked by removing it from the commitment system on the CCSM utilized by the BPI.

Test Passing Criteria:

The test will pass if the output pledge can only be used in the commercial value-creation/exchange event in which it was committed, and any attempt to use the same output pledge in another commercial value-creation/exchange event fails as long as said event is restricted to the CCSM the BPI uses for the Output Pledge. Additionally, the Output Pledge should be able to be revoked by removing it from the commitment system on the CCSM utilized by the BPI.

[CR20]<[O3]

A BPI MUST create an Output Pledge of a workstep output used in a commercial value-creation event as a succinct, non-interactive zero-knowledge proof of the pledge commitment (privacy preservation of an output pledged in a commercial value-creation event).

Note that an Output Pledge can contain, and usually will contain more than one workstep output used in a commercial value-creation event.

[CR20]<[O3] Testability:

Preconditions:

- The BPI test instance has a workstep output that will be used in a commercial value-creation event.
- The BPI is capable of generating an Output Pledge for the workstep output.
- The BPI utilizes a zero-knowledge proof system.

Test Steps:

1. Execute the workstep that generates the output to be used in a commercial value-creation event.
2. Generate an Output Pledge for the workstep output.
3. Verify that the generated zero-knowledge proof of the Output Pledge is non-interactive by checking that it does not require any interaction between the BPI and an external proof verifier.
4. Verify that the generated zero-knowledge proof of the Output Pledge is succinct by checking that the size of the proof in terms of computation steps is smaller than the number of computational steps required to solve the constraint equation system of the prover based on the zero-knowledge circuit used.

Test Passing Criteria:

- The generated zero-knowledge proof of the Output Pledge is non-interactive.
- The generated zero-knowledge proof of the Output Pledge is succinct.

[CR21]<[O3]

An Output Pledge MUST be committed to the CCSM utilized by the BPI (Liveness).

[CR21]<[O3] Testability: The same test as in [R229] Testability may be used.

[CR22]<[O3]

An Output Pledge MUST be verifiable by any 3rd party on the CCSM utilized by the BPI (censorship-resistant proof verifiability).

[CR22]<[O3] Testability: The same test as in [D29] Testability may be used.

[CR23]<[O3]

An Output Pledge MUST be updatable.

[CR23]<[O3] Testability: The same test as in [CR19]<[O3]](#cr19o3) Testability may be used since it involves an update operation of an Output Pledge.

[CR24]<[O3]

An Output Pledge MUST only be updated by the owners of the workstep output used in a commercial value-creation event.

Note that there are many ways this can be achieved. Note also that the ownership of the output might change and be highly fractionalized if used in a token as collateral. Therefore, there might be the need for a custodianship that is authorized to update the Output Pledge, and only under specific circumstances, such as when the outstanding token number is zero.

[CR24]<[O3] Testability:

Preconditions:

- A BPI test instance with the ability to create Output Pledges is set up and running.
- A workstep output has been used in a commercial value-creation event and an Output Pledge has been created for it.

Test Steps:

1. Verify that only the owners of the workstep output used in the commercial value-creation event are able to update the Output Pledge.
2. Attempt to update the Output Pledge using credentials of a non-owner of the workstep output.
3. Verify that the attempt to update the Output Pledge failed and the Output Pledge remained unchanged.
4. Attempt to update the Output Pledge using credentials of an owner of the workstep output.
5. Verify that the attempt to update the Output Pledge succeeded and the Output Pledge was updated accordingly.

Test Passing Criteria:

- The test is considered passed if the attempt to update the Output Pledge using credentials of a non-owner of the workstep output fails in Step 2, and the attempt to update the Output Pledge using credentials of an owner of the workstep output succeeds in Step 4, and the Output Pledge is updated accordingly in Step 5.

6.2 BPI Workflow

After specifying a workstep, this document will now turn to a workflow.

[R231]

A workflow MUST contain at least one workstep.

[R231] Testability:

Preconditions:

- A BPI test instance has been initialized and configured to support workflow creation.
- The BPI interface for creating workflows is accessible to a BPI Subject.

Test Steps:

Non-Standards Track Work Product

1. Create a new workflow using the BPI interface.
2. Verify that the workflow contains no workstep.
3. Attempt to save the workflow without any worksteps.
4. Verify that the BPI returns an error message and prevents the workflow from being saved.
5. Add a workstep to the workflow.
6. Verify that the BPI correctly saved the workstep to the workflow.

Test Passing Criteria:

- If the BPI returns an error message and prevents the workflow from being saved without any worksteps.
- If the BPI correctly saves a workflow with one workstep.

[R232]

If there is more than one workstep in a workflow, the worksteps in a workflow MUST be causally connected.

This means that the output of a workstep in a workflow is a required input into one or more subsequent worksteps.

[R232] Testability:

Preconditions:

- A BPI test instance has been initialized and configured to support workflow creation.
- A workflow has been created with more than one workstep.
- The input to the first workstep in the workflow has been defined.

Test steps:

1. Execute the first workstep in the workflow.
2. Verify that the output of the first workstep is used as input in the second workstep.
3. Repeat step 2 for all pairs of consecutive worksteps in the workflow.

Test Passing criteria:

- All pairs of consecutive worksteps in the workflow use the output of one workstep as input in the next workstep.

[D33]

If there is more than one workstep in a workflow, the zero-knowledge proof of correctness of the input to the last workstep in the workflow SHOULD be a proof that accumulates the valid proofs for all previous inputs.

Such proofs are known as inductive proof chains as each proof accumulates the previous proof, with further local inputs, in the causal chain of worksteps. The verifier, therefore, does not have to verify all proofs and hold all inputs but rather only the final one with the final public input to prove the validity of the entire computational chain.

[D33] Testability:

Preconditions:

- A BPI test instance has been initialized and configured to support workflow creation.
- A workflow has been created with more than one workstep.
- The input to the first workstep in the workflow has been defined.

Test Steps:

1. Create a workflow with at least two worksteps.
2. Generate valid zero-knowledge proofs of correctness of the inputs to all worksteps except the last one.
3. Generate an inductive proof chain, which accumulates all previous proofs, for the input to the last workstep.
4. Verify the inductive proof chain by checking if the final proof is valid.
5. Modify any of the inputs to any of the previous worksteps.
6. Regenerate the corresponding proof and update the inductive proof chain.
7. Verify the updated inductive proof chain by checking if the final proof is still valid.

Test Passing Criteria:

- The final proof of the inductive proof chain is valid.
- The inductive proof chain can be updated with the latest proof of any previous input without invalidating the final proof.

[D34]

The prover system in a workflow with more than one workstep SHOULD be the same for all worksteps.

[D34] Testability:

Preconditions:

- A BPI test instance has been initialized and configured to support workflow creation.
- A workflow with more than one workstep has been defined.
- Each workstep in the workflow has its own prover system for generating and verifying proofs.

Test Steps:

1. Select two worksteps in the workflow.
2. Generate a proof for the output of the first workstep using its prover system.
3. Verify the proof generated in step 2 using the prover system of the second workstep.
4. Record whether the verification in step 3 was successful or not.
5. Repeat steps 2-4 for all combinations of worksteps in the workflow.
6. Verify that for each combination of worksteps, the proof generated in one workstep can be verified using the prover system of another workstep.
7. If any verification in step 6 fails, record the specific combination of worksteps and the result of the verification.

Test Passing Criteria:

- All verifications in step 6 pass, indicating that the prover system in the workflow is the same for all worksteps.
- If any verification in step 6 fails, the test fails and the specific combination of worksteps and verification results are recorded for further investigation.

[R233]

A workflow with more than one workstep MUST have a unique identifier within a BPI.

Non-Standards Track Work Product

[R233] Testability:

Preconditions:

- A BPI test instance has been created.
- The BPI has at least two workflows.

Test steps:

1. Retrieve the list of workflows within the BPI.
2. For each workflow, retrieve its identifier.
3. Verify that each identifier is unique within the BPI.
4. Repeat steps 2 and 3 for all workflows in the BPI.
5. Verify that there are no duplicate identifiers within the BPI.
6. Attempt to change the identifier of one workflow to be the same as the identifier of another workflow.
7. Verify that the BPI does not allow this change and generates an appropriate error message.

Test passing criteria:

- All existing workflows in the BPI have a unique identifier.
- An attempt to assign the same identifier to two workflows fails and the BPI generates an appropriate error message.

[R234]

A workflow with more than one workstep and a given set of inputs **MUST** be sequentially executed by a BPI.

This simply means that for a given set of inputs there is only one process path through a given workflow.

[R234] Testability:

Preconditions:

- A BPI test instance is set up and configured.
- The BPI has at least one workflow with more than one workstep and a defined set of inputs.
- The inputs for the workflow are available and valid.

Test steps:

1. Start the BPI and provide the inputs for the workflow.
2. Verify that the BPI executes the first workstep in the workflow.
3. Verify that the output from the first workstep is used as input for the second workstep in the workflow.
4. Repeat steps 2 and 3 until all worksteps in the workflow have been executed.
5. Verify that the output of the last workstep in the workflow matches the expected output based on the provided inputs.

Test passing criteria:

- All worksteps in the workflow are executed in the correct order.
- The output of each workstep is used as input for the subsequent workstep.
- The final output of the workflow matches the expected output based on the provided inputs.

6.3 BPI Workgroup

In this section, the document will discuss the requirements for a workgroup. Note that which BPI Subjects may or may not be able to create a workgroup is up to the individual implementations. However,

[R235]

There **MUST** be at least one BPI Subject role that has the authorization to create a workgroup.

[R235] Testability:

Preconditions:

- A BPI test instance has at least one BPI Subject role defined.
- The BPI Subject has access to the BPI.

Test Steps:

1. Log in to the system as the user.
2. Verify that there is at least one BPI Subject defined in the system.
3. Attempt to create a workgroup using the BPI Subject's credentials.
4. Verify that the workgroup is successfully created.
5. Verify that the BPI Subject has the appropriate authorization to create a workgroup by attempting to create a workgroup using credentials that do not have the required authorization.
6. Verify that the BPI denies the attempt to create a workgroup using unauthorized credentials.

Test Passing Criteria:

- Step 2: At least one BPI Subject role is found in the system.
- Step 4: The workgroup is created successfully.
- Step 6: The system denies the attempt to create a workgroup using unauthorized credentials.

[R236]

A workgroup **MUST** consist of at least one BPI Subject participant.

Note that a workgroup participant may be a BPI Subject of another BPI than the BPI the workgroup is defined in. The appropriate authentication and authorization policies to enable such a scenario are up to implementers and beyond the scope of this document.

[R236] Testability:

Preconditions:

- A BPI Subject has access to a BPI test system with credentials that allow the creation of a workgroup.

Test steps:

1. Create a new workgroup.
2. Verify that the workgroup does not contain any participants.

Non-Standards Track Work Product

3. Add a BPI Subject participant to the workgroup.
4. Verify that the participant is added successfully.
5. Attempt to remove the participant from the workgroup.
6. Verify that the participant cannot be removed.
7. Attempt to add another participant to the workgroup.
8. Verify that the participant is added successfully.

Test passing criteria:

- Steps 1 to 4 should execute successfully.
- Step 5 should fail and raise an error or exception.
- Steps 6 to 8 should execute successfully.

[R237]

A workgroup MUST have at least one administrator.

[R237] Testability:

Preconditions:

- A workgroup exists in the BPI test system.
- The BPI Subject is logged in with the appropriate authorization to manage the workgroup.

Test Steps:

1. Add a new participant to the workgroup.
2. Verify that the participant has been added to the workgroup.
3. Remove the administrator from the workgroup.
4. Verify that the administrator has been removed and there are no other administrators in the workgroup.
5. Attempt to perform an administrative action, such as adding or removing a participant, as a non-administrator.
6. Verify that the action is not allowed and an appropriate error message is displayed.

Test Passing Criteria:

- Step 2: The participant should be successfully added to the workgroup.
- Step 4: The administrator should be successfully removed, and there should be no other administrators in the workgroup.
- Step 6: The action should not be allowed, and an appropriate error message should be displayed.

[R238]

A workgroup MUST have at least one security policy.

Note that a security policy consists of authentication and authorization rules for the workgroup participants. Note also that one or more workgroup administrators define the workgroup security policy.

[R238] Testability:

Preconditions:

- A workgroup exists in the BPI test system.
- A BPI Subject is logged in as an administrator of a workgroup.

Test steps:

1. Add a new security policy to the workgroup.
2. Verify that the security policy includes authentication rules that define the allowed authentication methods for workgroup participants.
3. Verify that the security policy includes authorization rules that define the allowed actions for workgroup participants.
4. Verify that the security policy is enforced for all workgroup participants.
5. Attempt to add a participant to the workgroup without specifying authentication credentials that satisfy the security policy.
6. Verify that the participant is not added.
7. Attempt to perform an unauthorized action in the workgroup as a participant.
8. Verify that the action is denied.

Test passing criteria:

- The security policy is added successfully.
- The security policy includes authentication and authorization rules.
- The security policy is enforced for all workgroup participants.
- The participant is not added if authentication credentials do not satisfy the security policy.
- The unauthorized action is denied for participants that do not have the required authorization according to the security policy.

[R239]

A workgroup MUST have at least one privacy policy.

A privacy policy consists of the data visibility rules for each participant.

[R239] Testability:

Preconditions:

- A workgroup exists in the BPI test system.
- A BPI Subject has the appropriate authorization to create a privacy policy for the workgroup.

Test steps:

1. Create a new privacy policy for the workgroup that restricts data visibility such that Participant A cannot see that Participant B is in the workgroup.
2. Add Participant A and Participant B to the workgroup.
3. Verify that Participant A cannot see that Participant B is in the workgroup.
4. Verify that Participant B cannot see that Participant A is in the workgroup.
5. Verify that a privacy policy has been created and associated with the workgroup.
6. Verify that if the privacy policy is attempted to be removed, the action fails and an appropriate error message is displayed.
7. Create a second privacy policy.
8. Verify that the second privacy policy can be removed.

Passing criteria:

Non-Standards Track Work Product

- The new privacy policy is successfully created.
- Participants cannot see other participants in the workgroup as specified by the privacy policy.
- The privacy policy is associated with the workgroup.
- The privacy policy cannot be removed from the group, if there is only one privacy policy defined for the workgroup.
- A second privacy policy is successfully created.
- The second privacy policy is successfully removed.

[R240]

A workgroup administrator **MUST** be able to perform at minimum the following functions:

- add or remove one or more participants
- create, update and delete both security and privacy policies.
- delete or archive a workgroup

Archiving a workgroup in the context of this document means that a workgroup cannot be actively used anymore. However, the workgroup data structures and associated data are accessible at any time but only by the participants of the archived workgroup.

[R240] Testability:

Preconditions:

- A BPI Subject is logged in as a workgroup administrator in a BPI test instance.
- The workgroup exists in the BPI.

Test steps:

- Add a new participant to the workgroup.
 - Verify that the participant has been added to the workgroup.
- Remove a participant from the workgroup.
 - Verify that the participant has been removed from the workgroup.
- Create a new security policy.
 - Verify that the security policy has been created and associated with the workgroup.
- Update an existing security policy.
 - Verify that the security policy has been updated and the changes are reflected in the workgroup.
- Delete an existing security policy.
 - Verify that the security policy has been deleted and is no longer associated with the workgroup.
- Create a new privacy policy.
 - Verify that the privacy policy has been created and associated with the workgroup.
- Update an existing privacy policy.
 - Verify that the privacy policy has been updated and the changes are reflected in the workgroup.
- Delete an existing privacy policy.
 - Verify that the privacy policy has been deleted and is no longer associated with the workgroup.
- Archive the workgroup.
 - Verify that the workgroup has been archived and is no longer accessible.

Test passing criteria:

- The participant has been added/removed from the workgroup as expected.
- The security/privacy policy has been created/updated/deleted as expected.
- The workgroup has been archived as expected.
- All test steps passed without any errors.

[O4]

A workgroup **MAY** have more than one administrator.

[O4] Testability: Execute the test in [R237] Testability twice.

[CR25]>[O4]

There **MUST** be a consensus model for administrative actions.

[CR25]<[O4] Testability:

Preconditions:

- A BPI test system is installed and running.
- A workgroup exists with at least one participant and an administrator.
- The consensus model for administrative actions is configured in the BPI system.

Test Steps:

1. The administrator creates a new security policy for the workgroup.
2. Another administrator attempts to delete the security policy created in step 1.
3. The system should prompt the second administrator to obtain consensus from the first administrator before proceeding with the deletion.
4. The second administrator obtains consensus from the first administrator and proceeds with the deletion.
5. The system should successfully delete the security policy.
6. The administrator adds a new participant to the workgroup.
7. Another administrator attempts to remove the participant added in step 6.
8. The system should prompt the second administrator to obtain consensus from the first administrator before proceeding with the removal.
9. The second administrator obtains consensus from the first administrator and proceeds with the removal.
10. The system should successfully remove the participant.
11. The administrator archives the workgroup.
12. Another administrator attempts to delete the workgroup.
13. The system should prompt the second administrator to obtain consensus from the first administrator before proceeding with the deletion.
14. The second administrator obtains consensus from the first administrator and proceeds with the deletion.
15. The system should successfully delete the workgroup.

Test Passing Criteria:

- The security policy is successfully created and deleted with the consensus of both administrators.
- The new participant is successfully added and removed with the consensus of both administrators.
- The workgroup is successfully archived and deleted with the consensus of both administrators.

[O5]

Non-Standards Track Work Product

A workgroup MAY be attached to one or more workstep instances.

[O5] Testability:

Preconditions:

- A BPI test system is set up and running.
- At least one workgroup is created.
- At least one workstep instance is available.

Test Steps:

1. Attach the workgroup to a workstep instance.
2. Verify that the workgroup is successfully attached to the workstep instance.
3. Detach the workgroup from the workstep instance.
4. Verify that the workgroup is successfully detached from the workstep instance.
5. Attach the workgroup to multiple workstep instances.
6. Verify that the workgroup is successfully attached to all the specified workstep instances.
7. Detach the workgroup from all the attached workstep instances.
8. Verify that the workgroup is successfully detached from all the attached workstep instances.

Test Passing Criteria:

- Each step in the test plan is executed successfully.
- The workgroup is successfully attached and detached from the workstep instances as specified.
- The workgroup is attached to multiple workstep instances simultaneously without any issues.

[CR26]>[O5]

A workgroup attached to a workflow MUST be also attached to each workstep in the workflow.

[CR26]<[O5] Testability:

Preconditions:

- A workflow with more than one workstep exists in a BPI test system.
- A workgroup is created in the BPI.

Test Steps:

1. Attach the created workgroup to the workflow.
2. For each workstep in the workflow:
 - Verify that the workgroup is attached to the workstep.
 - If the workgroup is not attached to the workstep, attach it to the workstep.
3. Detach the workgroup from the workflow.
4. Verify that the workgroup is detached from all worksteps in the workflow.
 - If the workgroup is still attached to any workstep, detach it from that workstep.

Test Passing Criteria:

- The workgroup is successfully attached to each workstep in the workflow.
- The workgroup is successfully detached from all worksteps in the workflow when detached from the workflow.

6.4 BPI Account

Before moving on to the requirements on the individual components in the processing layer required for agreement execution, this document needs to define and specify the prerequisites. Since this document has been defining and discussing state objects in the context of a BPI, it needs to define stateful object processing. This necessitates an account-based model for both BPI subjects and BPI state objects. This is analogous to the Ethereum model using accounts for individual participants and smart contracts – both are stateful objects. For simplicity, this document refers to account instead of BPI Account

[R241]

An account MUST be associated with one or more BPI Subjects.

[R241] Testability:

Preconditions:

- A BPI test system is running properly.
- A user has appropriate authorization to create and manage BPI subjects.
- There are no existing accounts associated with any BPI subjects.

Test Steps:

1. Create a new account in the BPI system.
2. Associate the account with a BPI subject.
3. Verify that the account is successfully associated with the BPI subject.
4. Associate the account with another BPI subject as in Step 2.
5. Verify that the account is successfully associated with the second BPI subject.
6. Attempt to associate an account with a non-existent BPI subject.
7. Verify that the system throws an error message indicating that the BPI subject does not exist.

Test Passing Criteria:

- Steps 1-5 must pass without any errors, and the account should be successfully associated with the BPI subject.
- Step 7 must result in an error message indicating that the BPI subject does not exist.

[R242]

A BPI State Object MUST be associated with an account.

[R242] Testability:

Preconditions:

- There is a BPI test system with at least one workflow that includes one or more worksteps.
- There is an account associated with the BPI.
- A workstep within the workflow produces a BPI State Object as output.

Non-Standards Track Work Product

Test Steps:

1. Access the account.
2. Navigate to the workflow within the BPI.
3. Check that there is at least one workstep that produces a BPI State Object as output.
4. Execute the workflow by providing the necessary inputs.
5. Check that the BPI State Object produced by the workstep is associated with the account.

Test Passing Criteria:

The test passes if the BPI State Object produced by the workstep is associated with the account.

An account itself is defined through the following requirements:

[R243]

An account **MUST** have a unique account identifier.

[R243] Testability:

Preconditions:

- There is an operational BPI test system

Test Steps:

1. Create two new accounts with account identifiers.
2. Verify that the account identifiers for each account are different.
3. Attempt to create a new account with an account identifier that already exists.
4. Verify that the system returns an error message indicating that the account identifier is already in use.
5. Delete one of the accounts.
6. Attempt to create a new account with the same account identifier as the deleted account.
7. Verify that the system allows the creation of the new account with the previously used account identifier.

Test Passing Criteria:

- Step 2: All account identifiers are unique.
- Step 4: The system returns an error message indicating that the account identifier is already in use.
- Step 7: The system allows the creation of the new account with the previously used account identifier.

[R244]

An account **MUST** have at least one account owner.

[R244] Testability:

Preconditions:

- A BPI test system is running and accessible.
- There are no existing accounts in the system.

Test Steps:

1. Create a new account with a BPI Subject in the owner role.
2. Verify that the account is created successfully with an account identifier.
3. Verify that the account has at least one account owner.
4. Create a new account with no account owner.
5. Verify that the BPI denies the account creation and creates an appropriate error message.

Test Passing Criteria: The test passes,

- If the account is created successfully with a unique account identifier and at least one account owner.
- If the BPI does not create an account if there is no BPI Subject indicated as the owner.

[O6]

An account **MAY** have more than one account owner.

[O6] Testability:

Preconditions:

- A BPI test system is running and accessible.
- There are no existing accounts in the system.

Test Steps:

1. Create a new account with a BPI Subject in the owner role.
2. Verify that the account is created successfully with an account identifier.
3. Verify that the account has at least one account owner.
4. Add a new account owner to the account.
5. Verify that the account has two account owners.

Test Passing Criteria: The test passes,

- If the account is created successfully with a unique account identifier and at least one account owner.
- If a second owner is successfully added to an account.

[CR27]>[O6]

If an account has more than one account owner, the account **MUST** have an account authorization condition.

An account authorization condition is a condition that has to be met by all or a portion of the account owners such as an m-of-n signature condition to authorize a BPI transaction from that account.

[CR27]<[O6] Testability:

Preconditions:

- A BPI test system is operational.

Non-Standards Track Work Product

- A user has appropriate authorization to create an account.
- An account has been created with multiple account owners.

Test Steps:

1. Log in to the BPI with appropriate authorization.
2. Navigate to the account details page of the account with multiple account owners.
3. Check if an account authorization condition is present.
4. Check that the account authorization condition is an m-of-n signature condition with a minimum number of signatures required is properly configured.
5. Add a new transaction request for the account.
6. Verify that the transaction request requires authorization from all account owners.
7. Log in to the BPI with one of the account owners.
8. Navigate to the transaction request and verify that it requires authorization from all account owners.
9. Provide authorization for the transaction request.
10. Repeat steps 7-9 for all account owners.
11. Verify that the transaction request is authorized only when the required number of account owners have provided their authorization.
12. Repeat steps 5-11 for different types of transaction requests.

Test Passing Criteria:

- Step 3 should show the presence of an account authorization condition.
- Step 4 should verify that the account authorization condition is an m-of-n signature condition with a minimum number of signatures required.
- Step 6 should verify that the transaction request requires authorization from all account owners.
- Steps 7-9 should successfully provide authorization for the transaction request for each account owner.
- Step 11 should verify that the transaction request is authorized only when the required number of account owners have provided their authorization.

[R245]

Account ownership authentication and account authorization conditions MUST be cryptographically provable.

This may be achieved through for example a cryptographic digital signature.

[R245] Testability:

Preconditions:

- A BPI test system is set up and running with one or more BPI Subjects and BPI Subject Accounts.
- One or more accounts exist.
- The BPI supports cryptographic algorithms.

Test Steps:

1. Log in to the BPI using the BPI Subject account.
2. Create an account, add BPI Subject as an owner, and create an account ownership authentication condition (e.g., a digital signature) for the account as a security policy.
3. Verify that the authentication condition is cryptographically provable by checking that the new account owner can be verified using the appropriate cryptographic algorithm and the BPI Subjects' public key when accessing the account.
4. Create an account authorization condition (e.g., an m-of-n signature condition) for the account.
5. Verify that the authorization condition is cryptographically provable by checking that it can be verified using the appropriate cryptographic algorithm and the public keys of the authorized BPI Subjects as owners by having one owner submit a BPI transaction.
6. Verify that the BPI checks the authentication and authorization conditions and allows the transaction only if they are valid.

Test Passing Criteria:

- The authentication and authorization conditions are created successfully.
- The authentication and authorization conditions are cryptographically provable.
- The BPI checks the authentication and authorization conditions during a transaction.
- The BPI allows the transaction only if the authentication and authorization conditions are valid.

[R246]

An account MUST have a deterministic nonce.

This ensures that transactions originating from an account are processed in the correct order.

[R246] Testability:

Preconditions:

- An account exists in a BPI test system.
- The BPI has a deterministic and unique nonce implementation for accounts.

Test Steps:

1. Retrieve the initial nonce value of the account.
2. Perform an action that triggers a nonce update, such as initiating a transaction.
3. Retrieve the updated nonce value of the account.
4. Repeat steps 2-3 multiple times, performing the same action each time.
5. Verify that the updated nonce values obtained in step 3 are unique and different from the initial nonce value and each other.
6. Verify that the nonce values used in step 4 can be independently recalculated using the nonce implementation in the BPI.

Test Passing Criteria:

- The initial nonce value is retrieved successfully.
- The updated nonce values obtained after performing the same action multiple times (steps 3-5) are unique and different from the initial nonce value, and each other.
- The updated nonce values from step 4. can be independently recalculated using the nonce implementation in the BPI.

[D35]

An account SHOULD have one or more units of value-accounting balances.

Also often known as tokens, these units of value-accounting allow the usage of state objects in financial transactions requiring units of accounting.

[D35] Testability:

Preconditions:

- A BPI test system is operational.
- An account is created and has one value-accounting balance.

Non-Standards Track Work Product

Test Steps:

1. Ensure that the account is created successfully.
2. Check if the account has one value-accounting balance.
3. Attempt to remove the value-accounting balance from the account.
4. Confirm that the removal of the value-accounting balance is not allowed (since it is recommended to have at least one balance).
5. Add a new value-accounting balance to the account.
6. Validate that the value-accounting balance is added to the account successfully.

Passing Criteria:

- The account has one value-accounting balance.
- The units of value-accounting balances are correctly associated with the account.
- The removal of the value-accounting balance is not allowed (since it is recommended to have at least one balance).
- The addition of a new value-accounting balance is successful.

[R247]

The state of an account MUST be represented by a cryptographic vector commitment scheme.

A cryptographic vector commitment scheme commits to an ordered sequence of q values (m_1, \dots, m_q) in such a way that one can later open the commitment at specific positions (e.g., prove that m_i is the i^{th} committed message). For security, Vector Commitments are required to satisfy position binding which states that an adversary should not be able to open a commitment to two different values at the same position. An example of such a scheme is a Merkle Trie.

[R247] Testability:

Preconditions:

- An account with an account state exists on a BPI test system.
- The chosen vector commitment scheme for the account state is a Merkle tree implementation.
- The initial account state is set up.

Test Steps:

1. Retrieve the initial account state from the BPI.
2. Perform operations that modify the account state:
 - a. Add a new transaction to the account.
 - b. Change the account settings.
3. Store the updated account state.
4. Reconstruct the updated account state Merkle tree using the updated account state values.
5. Generate the updated Merkle root hash from the reconstructed Merkle tree.
6. Retrieve the stored Merkle root hash from the updated account.
7. Compare the reconstructed Merkle root hash with the stored Merkle root hash.

Passing Criteria:

- If the Merkle root hashes in step 7. match the test succeeds, otherwise, it fails.

[R248]

Account properties consisting of more than one element MUST be represented by the same cryptographic vector commitment scheme as the full account and its state.

[R248] Testability:

Preconditions:

- An account with an account state exists on a BPI test system.
- The chosen vector commitment scheme for the account state is a Merkle tree implementation.
- The initial account state is set up.
- One account property is an object consisting of multiple elements.

Test Steps:

1. Retrieve the initial account state from the BPI.
2. Perform operations that modify the account state:
 - a. Update an element in the one multi-element account property.
3. Store the updated account state.
4. Reconstruct the updated account property Merkle tree using the updated account property element value.
5. Reconstruct the updated account state Merkle tree using the updated account state values taking into account that the value of the updated multi-element account property in the account state is the root hash of the updated Merkle tree of the multi-element account property.
6. Generate the updated Merkle root hash from the reconstructed Merkle tree.
7. Retrieve the stored Merkle root hash from the updated account.
8. Compare the reconstructed Merkle root hash with the stored Merkle root hash.

Passing Criteria:

- If the Merkle root hashes in step 8. match the test succeeds, otherwise it fails.

[R247] and [R248] ensure the structural integrity and cryptographic verifiability of the account at all times.

[R249]

The history of the state of an account MUST be represented by a cryptographic vector commitment.

This is required because not only does each state have to have structural integrity at all times but also its history with a causal connection between states.

[R249] Testability:

Preconditions:

- An account with an account state exists on a BPI test system.
- The chosen vector commitment scheme for the account state is a Merkle tree implementation.
- The initial account state is set up.
- The history of the above account exists on the same BPI test system.
- The chosen vector commitment scheme for the account state history is a Sparse Merkle tree implementation of the Merkle root hashes of the account state.
- The initial account state history is set up.
- The BPI test system can generate a Merkle Proof of Set Membership

Non-Standards Track Work Product

Test Steps:

1. Retrieve the initial account state from the BPI.
2. Perform operations that modify the account state:
 - Add a new transaction to the account.
 - Change the account settings.
3. Store the updated account state.
4. Retrieve the stored Merkle root hash from the updated account.
5. Retrieve the last Merkle root of the account state history
6. Generate a Merkle Proof of Set Membership of the stored Merkle root hash from the updated account.
7. Verify the Merkle Proof of Set Membership of the stored Merkle root hash from the updated account using the last Merkle root of the account state history and the stored Merkle root hash from the updated account.

Passing Criteria:

- If the Merkle proof successfully verifies, the test succeeds, otherwise it fails.

[R250]

The state of an account MUST be minimally comprised of the following elements:

- Account Identifier
- Owner(s)
- Authorization Condition (if more than one owner)
- Account Nonce
- State Object Prover System representation (if an account is associated with a state object)
- State Object storage (if an account is associated with a state object)

[R250] Testability:

Preconditions:

- The account system in a test BPI instance is set up and operational.
- The necessary components for representing the account state are implemented.

Test Steps:

1. Create a new account.
2. Verify that the account has the following elements:
 - Account Identifier
 - Owner(s)
 - Authorization Condition (if more than one owner)
 - Account Nonce
 - State Object Prover System representation (if associated with a state object)
 - State Object storage (if associated with a state object)
3. Modify each of the elements individually and verify that the changes are reflected in the account state.
4. Ensure that the account state remains consistent throughout the modifications.
5. Perform account operations such as transactions or updates and verify that the account state is updated accordingly.
6. Validate that the account state can be retrieved accurately and all the required elements are present.
7. Test the account state with multiple owners and verify that the authorization condition is correctly enforced.

Passing criteria:

- The account state contains all the required elements mentioned in the requirement.
- The modifications to individual elements are correctly reflected in the account state.
- The account state remains consistent throughout the operations and modifications.
- The authorization condition (if applicable) is properly enforced for accounts with multiple owners.

[R251]

The state of an account MUST only be changed based on a valid transaction compliant with the account authorization condition(s).

[R251] Testability:

Preconditions:

- The account system and state object functionality are set up and operational in a BPI test instance.
- An account with authorization condition(s) and associated state object exists.
- The BPI supports creating and processing transactions.

Test Steps:

1. Create a new transaction that attempts to modify the state of the associated state object.
2. Verify that the transaction includes all the required information, such as the account identifier, transaction details, and any additional data required by the authorization condition(s).
3. Check if the transaction complies with the authorization condition(s) defined for the associated account.
4. If the transaction is compliant with the authorization condition(s):
 - Apply the transaction to update the state of the associated state object.
 - Verify that the state object's state has been modified as expected.
5. If the transaction is not compliant with the authorization condition(s):
 - Reject the transaction and ensure that the state of the associated state object remains unchanged.
6. Repeat steps 2-6 with different types of transactions and authorization conditions, including valid transactions and transactions that do not meet the authorization condition(s).
7. Verify that only valid transactions compliant with the authorization condition(s) successfully modify the state of the associated state object.

Passing criteria:

- Valid transactions compliant with the authorization condition(s) successfully modify the state of the associated state object.
- Transactions that do not meet the authorization condition(s) are rejected, and the state of the associated state object remains unchanged.
- The BPI accurately enforces the authorization condition(s) to control state object modifications.

[R252]

The state of a state object MUST only be changed based on a valid transaction compliant with the authorization condition(s) of the account to which the state object belongs.

This document will discuss the requirements of a transaction and what constitutes a valid transaction in the next section. Note, that an account is associated with a workflow instance through the shared state objects.

Non-Standards Track Work Product

[R252] Testability:

Preconditions:

- A BPI test system is set up and operational.
- The account system and state object functionality are implemented.
- An account with associated state object(s) exists.
- The authorization condition(s) for the account and state object(s) are properly defined.
- Valid transaction types and corresponding authorization conditions are implemented.

Test Steps:

1. Create a valid transaction to modify the state of a state object associated with an account.
 - Ensure the transaction includes all required information, such as the account identifier, transaction details, and any additional data required by the authorization condition(s).
2. Verify that the transaction complies with the authorization condition(s) defined for the associated account and state object.
 - Validate that the transaction meets the necessary criteria specified in the authorization condition(s), such as signatures or other authentication mechanisms.
3. If the transaction is compliant with the authorization condition(s):
 - Apply the transaction to update the state of the associated state object.
 - Ensure the state object's state has been modified as expected.
4. If the transaction is not compliant with the authorization condition(s):
 - Reject the transaction and ensure that the state of the associated state object remains unchanged.

Passing Criteria:

- Only valid transactions compliant with the authorization condition(s) successfully modify the state of the associated state object.
- Transactions that do not meet the authorization condition(s) are rejected, and the state of the associated state object remains unchanged.
- The BPI system accurately enforces the authorization condition(s) to control state object modifications.
- The integrity and security of the state object's state are maintained, ensuring that unauthorized modifications are prevented.

6.5 BPI Transactions

Account states, and therefore, the state of BPI state objects, and, thus, agreement states are altered through BPI transactions submitted by requesters of state changes from their accounts. In the following, this document specifies requirements for the structure and characteristics of transactions.

[R253]

Each transaction MUST minimally have the following identifiers:

- Workflow (instance) ID (UID)
- Workstep (instance) ID (UID)
- Transaction ID (UID)

Note that a Workflow (Instance) ID may be the same as a Workstep (Instance) ID, if the workflow has only one workstep.

[R253] Testability:

Preconditions:

- A BPI test system is set up and operational.
- The workflow and workstep functionality are implemented.
- The generation of unique identifiers (UIDs) is in place.

Test Steps:

1. Create a transaction for a workflow with multiple worksteps.
 - Generate a unique Workflow ID (UID) for the transaction.
 - Generate a unique Transaction ID (UID) for the transaction.
 - Generate unique Workstep IDs (UIDs) for each workstep involved in the transaction.
2. Verify that the transaction includes the required identifiers.
 - Ensure that the Workflow ID, Transaction ID, and Workstep IDs are correctly associated with the transaction.
3. If the workflow has only one workstep:
 - Confirm that the Workflow ID and Workstep ID are the same.
 - Validate that the Transaction ID is unique.
4. If the workflow has multiple worksteps:
 - Ensure that the Workflow ID and Transaction ID are unique.
 - Verify that each Workstep ID is unique and associated with the corresponding workstep in the workflow.

Passing Criteria:

- Each transaction must have a valid and unique Workflow ID (UID), Workstep ID (UID), and Transaction ID (UID).
- For workflows with only one workstep, the Workflow ID and Workstep ID must be the same, while the Transaction ID remains unique.
- For workflows with multiple worksteps, each workstep must have a unique Workstep ID associated with the corresponding transaction and workflow.
- The BPI system correctly handles the generation and assignment of identifiers, ensuring consistency and uniqueness.
- The identifiers enable proper tracking, identification, and correlation of transactions within the BPI system.

[R254]

The Workflow Instance ID MUST be derivable from the Workflow ID.

[R254] Testability:

Preconditions:

- A BPI test system is set up and operational.
- The workflow functionality is implemented.
- The Workflow ID generation and derivation logic are in place.

Test Steps:

1. Generate a Workflow ID.
 - Create a new workflow and generate a unique Workflow ID.
2. Derive the Workflow Instance ID from the Workflow ID.
 - Apply the derivation logic to generate the Workflow Instance ID based on the Workflow ID.
3. Verify that the Workflow Instance ID is correctly derived.
 - Compare the derived Workflow Instance ID with the expected Workflow Instance ID based on the Workflow ID.
4. Create another Workflow ID.
 - Generate a new unique Workflow ID for a different workflow.

Non-Standards Track Work Product

5. Derive the Workflow Instance ID for the second workflow.
 - Apply the derivation logic to generate the Workflow Instance ID based on the second Workflow ID.
6. Verify that the second Workflow Instance ID is correctly derived.
 - Compare the derived Workflow Instance ID with the expected Workflow Instance ID based on the second Workflow ID.

Passing Criteria:

- The Workflow Instance ID must be correctly derived from the Workflow ID.
- The derived Workflow Instance ID must be unique and consistent for each corresponding Workflow ID.
- The derivation logic ensures a deterministic and consistent mapping between the Workflow ID and the Workflow Instance ID.
- The BPI system correctly applies the derivation logic and generates accurate Workflow Instance IDs.

[R255]

The Workstep Instance ID MUST be derivable from the Workstep ID.

[R255] Testability:

Preconditions:

- A BPI test system is set up and operational.
- The workflow and workstep functionality is implemented.
- The Workstep ID generation and derivation logic are in place.

Test Steps:

1. Generate a Workstep ID.
 - Create a new workstep and generate a unique Workstep ID.
2. Derive the Workstep Instance ID from the Workstep ID.
 - Apply the derivation logic to generate the Workstep Instance ID based on the Workstep ID.
3. Verify that the Workstep Instance ID is correctly derived.
 - Compare the derived Workstep Instance ID with the expected Workstep Instance ID based on the Workstep ID.
4. Create another Workstep ID.
 - Generate a new unique Workstep ID for a different workstep.
5. Derive the Workstep Instance ID for the second workstep.
 - Apply the derivation logic to generate the Workstep Instance ID based on the second Workstep ID.
6. Verify that the second Workstep Instance ID is correctly derived.
 - Compare the derived Workstep Instance ID with the expected Workstep Instance ID based on the second Workstep ID.

Passing Criteria:

- The Workstep Instance ID must be correctly derived from the Workstep ID.
- The derived Workstep Instance ID must be unique and consistent for each corresponding Workstep ID.
- The derivation logic ensures a deterministic and consistent mapping between the Workstep ID and the Workstep Instance ID.
- The BPI system correctly applies the derivation logic and generates accurate Workstep Instance IDs.

[D36]

A Transaction ID SHOULD be generated by the transaction originator/sender.

[D36] Testability:

Preconditions:

- A BPI test system is set up and operational.
- The transaction functionality is implemented.
- The system allows for the generation of Transaction IDs.

Test Steps:

1. Initiate a transaction.
 - The transaction originator/sender initiates a transaction within the BPI system.
2. Generate a Transaction ID for the transaction.
 - The BPI system generates a unique Transaction ID for the initiated transaction.
3. Verify that the Transaction ID is generated by the originator/sender.
 - Check whether the generated Transaction ID is provided by the transaction originator/sender.
4. Perform multiple transactions by different originators/senders.
 - Repeat steps 1 to 3 for multiple transactions initiated by different originators/senders.

Passing Criteria:

- The BPI system successfully generates a unique Transaction ID for each initiated transaction.
- The Transaction ID is provided by the transaction originator/sender.
- Different originators/senders generate distinct Transaction IDs for their respective transactions.
- The generated Transaction IDs are consistently and accurately associated with their corresponding transactions.

[R256]

A Transaction MUST have a **From** (Sender) and a **To** (Receiver) element each containing the respective Sender and Receiver account numbers.

[R256] Testability:

Preconditions:

- A BPI test system is set up and operational.
- The transaction functionality is implemented.
- The system has established accounts with unique account numbers.

Test Steps:

1. Initiate a transaction.
 - The transaction initiator/sender starts the process of initiating a transaction within the BPI system.
2. Specify the Sender and Receiver account numbers.
 - The transaction initiator/sender provides the account number of the sender and receiver as part of the transaction details.
3. Verify the presence of the **From** and **To** elements.
 - Check whether the transaction includes the **From** element, which contains the sender's account number, and the **To** element, which contains the receiver's account number.
4. Validate the accuracy of the Sender and Receiver account numbers.

Non-Standards Track Work Product

- Confirm that the provided account numbers in the `From` and `To` elements match the intended sender and receiver accounts.
- 5. Perform multiple transactions with different sender and receiver accounts.
 - Repeat steps 1 to 4 for multiple transactions involving different sender and receiver accounts.

Passing Criteria:

- Each transaction must include the `From` and `To` elements.
- The `From` element should contain the correct account number of the sender.
- The `To` element should contain the correct account number of the receiver.
- The sender and receiver account numbers provided in the transactions must match the intended accounts.
- Different sender and receiver accounts should be successfully used in multiple transactions.

[R257]

Each transaction MUST have a different, deterministic nonce based on the account of the sender.

[R257] Testability:

Preconditions:

- A BPI test system is set up and operational.
- There exists an account with a unique account number.
- There are no existing transactions associated with the account.

Test Steps:

1. Generate the first transaction with a unique sender account number.
2. Record the nonce of the first transaction.
3. Generate the second transaction with the same sender account number.
4. Record the nonce of the second transaction.
5. Repeat steps 3 and 4 for several transactions initiated by the same sender account.
6. Verify that each recorded nonce is different from the previous nonces.
7. Derive the nonce for each transaction using the same deterministic method as the sender account.
8. Compare the derived nonces with the recorded nonces for each transaction.
9. Verify that the derived nonces match the recorded nonces for all the transactions.

Passing Criteria:

- The recorded nonces for all the transactions initiated by the same sender account must be unique and different from each other.
- The derived nonces using the deterministic method must match the recorded nonces for all the transactions.

[R258]

Each transaction MUST contain a representation of a document as a state object constituting the suggested new agreement state.

[R258] Testability:

Preconditions:

- Set up a BPI test system with transaction capabilities.
- Create at least one sender account and one receiver account.

Test Steps:

1. Generate a unique Transaction ID.
2. Prepare a document representing the suggested new agreement state.
3. Create a transaction with the required field and attach the document as a state object.
4. Send the transaction to the BPI system for processing.

Passing Criteria:

- The transaction is successfully submitted to the BPI system without any errors or rejections.
- The BPI system validates and accepts the transaction with the document as a state object.
- The document is associated with the respective transaction.
- The BPI system updates the agreement state based on the contents of the document.

[R259]

A transaction MUST contain the cryptographic digital signature of one or more of the owner(s) of the Sender account compliant with the account's authorization condition.

[R259] Testability:

Preconditions:

- Set up a BPI test system with transaction capabilities.
- Create at least one sender account with multiple owners.
- Define the authorization condition for the sender account.

Test Steps:

1. Prepare the transaction data, including all the necessary fields
2. Generate a unique Transaction ID.
3. Compute the cryptographic digital signature(s) of one or more owners of the sender account, compliant with the account's authorization condition.
4. Include the digital signature(s) in the transaction data.
5. Send the transaction to the BPI system for processing.

Passing Criteria:

- The transaction is successfully submitted to the BPI system without any errors or rejections.
- The BPI system verifies the digital signature(s) against the sender account's authorization condition.
- If the signature(s) are compliant with the authorization condition, the BPI system accepts the transaction.
- If the signature(s) do not comply with the authorization condition, the BPI system rejects the transaction.

[R260]

A transaction MUST be considered invalid if one of the following conditions is met:

- The transaction nonce is not equal to the next deterministic value of the account nonce.

Non-Standards Track Work Product

- The cryptographic signature of the account owner(s) on the transaction cannot be verified
- The account authorization condition for the Sender account is not met.
- The transaction does not have an existing Workflow (Instance) ID, Workstep (Instance) ID
- The transaction From or To are not valid account identifiers
- The transaction does not contain a proposed new agreement state object.

[R260] Testability:

Preconditions:

- Set up a BPI test system with transaction capabilities.
- Create a Sender account with its corresponding account nonce, owner(s), and authorization condition.
- Establish a workflow with a unique Workflow (Instance) ID associated with a unique workstep with a Workstep (Instance) ID.
- Ensure the existence of valid account identifiers for From and To in the transaction.
- Prepare a proposed new agreement state object.

Test Steps:

1. Retrieve the next deterministic value of the account nonce.
2. Generate a transaction with the necessary fields, including the transaction nonce, cryptographic signature of the account owner(s), Workflow (Instance) ID, Workstep (Instance) ID, From and To account identifiers, and the proposed new agreement state object.
3. Verify that the transaction nonce is equal to the next deterministic value of the account nonce.
4. Verify the cryptographic signature of the account owner(s) on the transaction to ensure it can be successfully verified.
5. Check if the account authorization condition for the Sender account is met.
6. Verify the existence of Workflow (Instance) ID and Workstep (Instance) ID in the transaction.
7. Validate the From and To account identifiers to confirm their validity.
8. Ensure the presence of a proposed new agreement state object in the transaction.

Passing Criteria:

- The transaction is considered valid if all the conditions listed in the preconditions are met.
- The transaction nonce matches the next deterministic value of the account nonce.
- The cryptographic signature of the account owner(s) on the transaction is successfully verified.
- The account authorization condition for the Sender account is met.
- The transaction contains an existing Workflow (Instance) ID and Workstep (Instance) ID.
- The From and To account identifiers are valid and correctly specified in the transaction.
- The transaction includes a proposed new agreement state object.

If any of the above criteria are not met the test fails.

Note, that this is only a minimal set of requirements on an invalid transaction. Each BPI can define other requirements not covered above.

In the following, this document will discuss the transaction lifecycle and its requirements as it pertains to Agreement Execution.

6.6 BPI Transaction Lifecycle

The figure below showcases at a high level the flow of a transaction through a BPI. In the subsequent discussion on transaction lifecycle characteristics and requirements, this document will focus on the BPI Processing Layer and the steps indicated in blue.

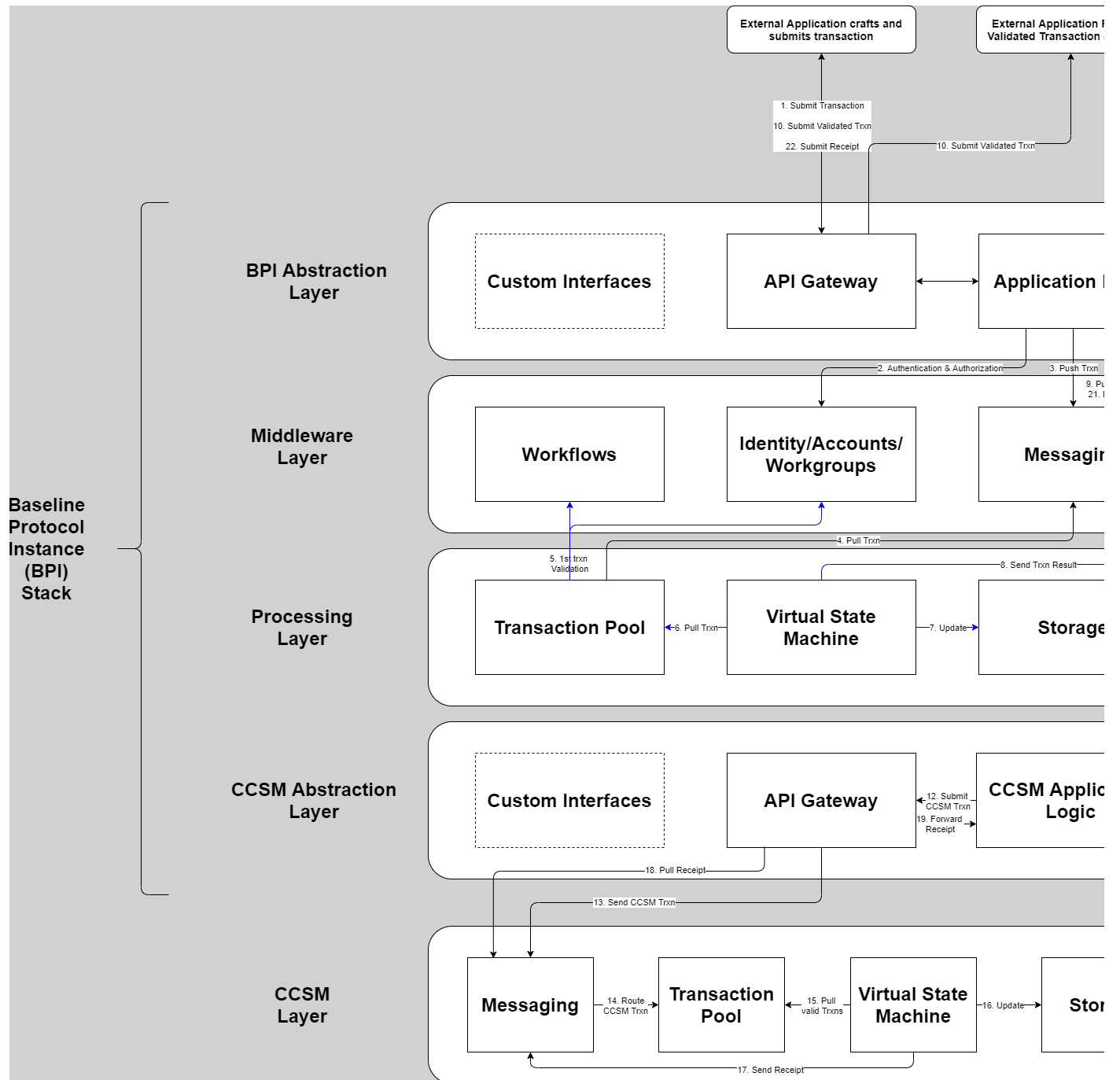


Figure 10: High Level Transaction Lifecycle Flow through the BPI

A prerequisite for transaction lifecycle step 4 where the BPI Processing Layer Transaction Pool pulls a transaction from a BPI Middleware Layer's Messaging is that there is at least one transaction in the Messaging Capability waiting to be processed by the BPI Processing Layer.

The requirements for the subsequent step (5) are as follows:

[R261]

A Transaction Pool MUST be able to validate all transaction requirements for a valid transaction in section [6.5 BPI Transactions](#).

[R261] Testability:

Preconditions:

- Set up a BPI test system with a Transaction Pool component.

Test Steps:

1. Generate a transaction with all required fields
2. Submit the transaction to the Transaction Pool for validation.
3. Verify that the Transaction Pool successfully validates the transaction.

Non-Standards Track Work Product

4. Confirm that the transaction is added to the valid transaction pool.
5. Generate a transaction missing one or more required fields.
6. Submit the transaction to the Transaction Pool for validation.
7. Verify that the Transaction Pool does not validate the transaction and responds with an appropriate error message.
8. Confirm that the transaction is not added to the valid transaction pool.

Passing Criteria:

- The Transaction Pool validates all required fields and transaction requirements of a transaction.
- The Transaction Pool successfully validates the first generated transaction without any errors or failures.
- The validated transaction is added to the valid transaction pool, indicating that it meets all the necessary requirements.
- The Transaction Pool rejects the second, invalid transaction and responds with an appropriate error message.
- The rejected Transaction is not in the valid transaction pool.

[R262]

A Transaction Pool MUST order transactions for processing based on the order of their unique messaging ID and their account nonces for the same 'From' account.

[R262] Testability:

Preconditions:

- Set up a BPI test system with a Transaction Pool component.

Test Steps:

1. Create two transactions with the same "From" account but different messaging IDs and nonces. Ensure that the messaging ID and nonces are ordered such that the second transaction has a higher messaging ID and nonce compared to the first transaction.
2. Submit the transactions to the Transaction Pool for processing.
Verify that the Transaction Pool correctly orders the transactions based on their unique messaging ID and account nonces.
3. Confirm that the first transaction is processed before the second transaction, indicating that the Transaction Pool follows the specified ordering criteria.

Passing Criteria:

- The Transaction Pool orders transactions for processing based on their unique messaging ID and account nonces.
- The Transaction Pool correctly orders the submitted transactions according to their messaging ID and nonces.
- The first transaction with a lower messaging ID and nonce should be processed before the second transaction with a higher messaging ID and nonce.

[R263]

If the order of two or more transactions from the same account in the Transaction Pool is in contradiction to the order of the deterministic account nonce, the transactions MUST first be ordered by their account nonces and then by their message IDs.

Note, that this is required to avoid a BPI operator ordering transactions maliciously, or injecting malicious transactions ahead of other transactions.

[R263] Testability:

Preconditions:

- Set up a BPI test system with a Transaction Pool component.

Test Steps:

1. Create three transactions from the same account with different message IDs and nonces.
2. Submit the transactions in a contradictory order, where the order of nonces and message IDs is not aligned.
3. Submit the transactions to the Transaction Pool for processing.
4. Verify that the Transaction Pool correctly handles the ordering of transactions based on the account nonces and message IDs.
5. Confirm that the transactions are reordered by the Transaction Pool following the specified requirement: first by the account nonces in ascending order, and then by the message IDs in ascending order.
6. Verify that the reordered transactions are processed in the correct order, i.e., the transaction with the lowest nonce first.

Passing Criteria:

- The Transaction Pool enforces the specified ordering based on account nonces and message IDs correctly.
- The Transaction Pool correctly reorders the transactions from the same account if their order contradicts the order of the deterministic account nonce.
- The reordered transactions are processed in the correct order according to the ordering criteria.

[R264]

A Transaction Pool MUST create a batch of transactions to be processed by the Virtual State Machine given one or more batching parameters such as time, number of transactions, or storage size of a batch.

The number of transactions per batch and the time period covered by each batch while fixed can be freely chosen by an implementer. Recommendations as to batch size and time frame will be given in an implementers guide and is beyond the scope of this document.

[R264] Testability:

Preconditions:

- Set up a BPI test system with a Transaction Pool component.
- Configure the Transaction Pool with batching parameters, such as time interval, number of transactions, or storage size.

Test Steps:

1. Initiate the BPI system and Transaction Pool.
2. Verify that the Transaction Pool has the specified batching parameters set.
3. Monitor the Transaction Pool to ensure it collects transactions for batching.
4. Check if the Transaction Pool creates a batch of transactions based on the specified batching parameters.
5. Confirm that the batch of transactions is passed to the Virtual State Machine for processing.

Passing Criteria:

- The Transaction Pool successfully collects transactions based on the specified batching parameters.
- A batch of transactions is created according to the specified batching parameters.
- The batch of transactions is correctly forwarded to the Virtual State Machine for processing.

[R265]

Non-Standards Track Work Product

A Transaction Pool MUST process an invalid transaction by assigning the transaction an error code and an easily human-readable error message and issuing a message minimally consisting of the tuple (Sender Account, Error Code, Error Message, Transaction) to the Messaging Capability of the BPI to inform the sender of the transaction failure and its reason.

[R265] Testability:

Preconditions:

- Set up a BPI test system with a Transaction Pool component.
- Messaging Capability of the BPI to communicate with the transaction sender.

Test Steps:

1. Initiate the BPI system and Transaction Pool.
2. Submit an invalid transaction to the Transaction Pool.
3. Verify that the Transaction Pool detects the invalidity of the transaction.
4. Verify that the transaction sender receives an error message containing the tuple (Sender Account, Error Code, Error Message, Transaction).

Passing Criteria:

- The Transaction Pool correctly detects and identifies invalid transactions.
- An error code and an easily human-readable error message are assigned to the invalid transaction.
- The message containing the tuple (Sender Account, Error Code, Error Message, Transaction) is successfully issued to the Messaging Capability.
- The sender receives the message from the Messaging Capability, informing them of the transaction failure and providing the reason for the failure.

In the following, this document will discuss the requirements on the Virtual State Machine of the BPI Processing Layer.

Since BPIs are used to verify the correctness of state transitions (see step (6) in Fig. 10 above), BPIs will utilize a Virtual State Machine (VSM) for its computations to validate state transitions of state objects; a digital computer running on a physical computer. A VSM requires architecture and execution rules which together define the Execution Framework.

[R266]

The Execution Framework of a VSM MUST be deterministic.

Any BPI running the same Execution Framework on the same state object with the same input data needs to arrive at the same result, in other words, deterministic outcomes. This is only guaranteed if the Execution Framework either does not allow instructions to be executed in parallel, but only strictly sequential, or if the Execution Framework has methods in place that allow the identification and prevention of transactions that would cause state conflicts if processed in parallel.

For example, the Buyer, also known as Requester, proposes a commercial state change of the MSA through Order A which is created at time t , and the Seller, also known as the Provider, has just agreed to a suggested discount rate change in the MSA submitted by the Buyer at time $t-1$ but not yet processed. This means that if the transaction of Order A is processed in parallel to the discount change the wrong discount might be applied to Order A depending on which transaction is executed first.

[R266] Testability:

Preconditions:

- Set up a BPI test system with a Virtual State Machine (VSM) component.
- Configure and initialize the Execution Framework of the VSM.

Test Steps:

1. Execute a series of transactions on the BPI system.
2. Record the sequence of actions and events within the Execution Framework during the transaction execution.
3. Repeat Step 1 multiple times with the same initial state and transactions.
4. Compare the recorded sequences of actions and events from Step 2 for each execution.
5. Verify that the recorded sequences of actions and events are identical for all executions.
6. Introduce variations in the execution environment, such as system load or resource availability.
7. Repeat Step 1 with the same initial state and transactions under the varied execution environment.
8. Compare the recorded sequences of actions and events from Step 2 for each execution under the varied environment.
9. Verify that the recorded sequences of actions and events remain identical despite the variations in the execution environment.

Passing Criteria:

- The recorded sequences of actions and events within the Execution Framework are identical across multiple executions with the same initial state and transactions.
- The recorded sequences of actions and events remain consistent even when the execution environment is varied.

[R267]

The Execution Framework of a VSM MUST ensure that state transition validation computations are either completed or aborted in finite time.

Note that what is deemed to be a suitable, finite time is determined by the allowable duration of a transaction. This requirement means that infinite computational loops cannot be allowed in a BPI.

[R267] Testability:

Preconditions:

- Set up a BPI test system with a Virtual State Machine (VSM) component.
- Configure and initialize the Execution Framework of the VSM.

Test Steps:

1. Initiate a state transition within the BPI system by submitting a transaction to the BPI.
2. Measure the elapsed time for the state transition validation computation triggered by the transaction to complete.
3. Repeat Step 1 with different state transitions, including both simple and complex scenarios.
4. Measure the elapsed time for each state transition validation computation.
5. Verify that the elapsed time for each state transition validation computation is finite and within an acceptable range.
6. Introduce a state transition that triggers a potentially infinite computation or an exceptionally long validation process by submitting a transaction to the BPI.
7. Ensure that the computation either completes within a predetermined time limit or set resource consumption limit or is aborted after exceeding the time or resource limit.
8. Repeat Steps 6-7 with different problematic state transitions, covering various edge cases.

Passing Criteria:

- The elapsed time for state transition validation computations triggered by a transaction is finite and within an acceptable range for all tested scenarios.
- Problematic state transitions triggering potentially infinite computations or exceptionally long validation processes are either completed within a predetermined time limit or resource consumption limit or aborted if the time or resource limit is exceeded.

[R268]

The Execution Framework of a VSM MUST support commonly used cryptographic primitives for zero-knowledge proofs, e.g., hashing, commitments, accumulators, or zero-knowledge proof verification.

Non-Standards Track Work Product

[\[R268\]](#) Testability:

Preconditions:

- Set up a BPI test system with a Virtual State Machine (VSM) component that supports a zero-knowledge proof framework such as zk-SNARK or zk-STARK
- A transaction that triggers a zero-knowledge proof generation event based in the VSM.
- A simple logic that verifies the integrity of a signed simple document translated into zero-knowledge circuit (public inputs: Roothash of the document mapped into a Merkle Tree, the public key that signed the document, Private Data: The Merkle Proofs for each data element in the document, the digital signature over the root hash of the document Merkle tree) that verifies that the Merkle proofs for all leafs are correct and that the digital signature of the root hash is also correct. This circuit tests the following common cryptographic primitives used in a zero-knowledge proofs: hashing, accumulation via the Merkle tree, digital signatures, zero-knowledge prover system.

Test Steps:

1. Transform the simple document into a Merkle tree and digitally sign the Merkle Tree root hash, and generate the Merkle Proofs for each document element as a leaf in the Merkle tree.
2. Submit the data as public and private input data to the zero-knowledge prover service of the VSM for one of the zero-knowledge prover systems supported by the service such as PLONK, FRI, or Groth16. This tests the cryptographic primitives used in the circuit arithmetization and commitment schemes utilized such as a Kate commitment scheme
3. Submit the zero-knowledge proof received from the VSM to the zero-knowledge verifier service of the VSM.
4. Verify that the zero-knowledge proof is correct.

Passing Criteria:

- The document is correctly transformed into a Merkle tree and its root hash correctly signed.
- The zero-knowledge proof of document integrity is correctly generated and is correctly verified.

[\[D37\]](#)

The Execution Framework of a VSM SHOULD have a mathematical proof of correctness and security.

[\[D37\]](#) Testability: There is a peer-reviewed academic paper that proves the mathematical correctness and security of the utilized VSM given certain security assumptions such as the hardness of the Diffie-Hellman Discrete Logarithm and framework such as the universal composability framework.

[\[D38\]](#)

The Execution Framework of a VSM SHOULD be Verifiably Secure.

[\[D38\]](#) Testability:

Preconditions:

- Set up a BPI test system with a Virtual State Machine (VSM) component.
- The necessary functions or computations that can be offloaded to other VSM clients within the Execution Framework are defined.
- A mechanism to verify the correctness of a computation offloaded from the main VSM to another VSM client within the Execution Framework is implemented in the Execution Framework.
- A protocol or mechanism for other clients to evaluate the function and return the result with a proof of correct computation is implemented.
- A proof of correct execution provided by a VSM client can be independently verified within the main VSM of the Execution Framework.
- A security analysis of the Execution Framework, focusing on the verifiably secure mechanism and the assumptions used in the proof of computation has been completed and any findings remediated including an assessment of the validity of the security assumptions used in the proof and their impact on the overall security of the Execution Framework, and an evaluation of the resilience of the Execution Framework against potential attacks or attempts to manipulate the computation results or the verifiable proof.
- A thorough review of the mathematical proofs and algorithms used in the Execution Framework, ensuring their correctness and reliability has been completed.

Test Steps:

1. Perform a series of test computations in the main VSM, offloading them to other VSM clients
2. Verify that the returned results can be verified as correct through the provided proofs by the main VSM.

Passing Criteria:

- The Execution Framework meets all preconditions.
- Other VSM clients can successfully evaluate the offloaded functions from the main VSM and provide a proof of correct computation.
- The Execution Framework independently verifies the provided proofs and validates the correctness of the computation results.

[\[R269\]](#)

If a VSM can generate a valid Proof-of-Correctness for a transaction, it MUST update the state and the state history of the state object the transaction targeted based on the transaction data.

[\[R269\]](#) Testability:

Preconditions:

- A BPI test system with a Virtual State Machine (VSM) component is running.
- BPI contains a test state object associated with an account connected to a workstep within a workflow connect to a workgroup.
- A BPI Subject participant in the workgroup.
- A representation of the expected updated state object and its updated history.

Test Steps:

1. Create a valid transaction with appropriate transaction data based on the defined workstep.
2. Submit the transaction to the BPI VSM via the workstep for processing.
3. Verify that the VSM generates a Proof-of-Correctness for the transaction.
4. Check if the VSM updated the state object associated with the account associated with the workstep.
5. Check if the VSM updated the state history of the targeted state object based on the transaction data.
6. Validate that the updated state object and the state history of the state object equal the previously generated representation of the state object and its history (initial state object value).

Passing Criteria:

- A valid transaction with appropriate transaction data is created.
- The valid transaction is submitted to the VSM for processing.
- The VSM generates a valid Proof-of-Correctness for the transaction.
- The VSM updates the state object and its history based on the transaction data.
- The updated state and state history align with the expected changes specified by the transaction data.

[\[R270\]](#)

A VSM MUST store all proofs, state objects, their associated data, and their histories in the Storage capability of the BPI Processing Layer.

[\[R270\]](#) Testability:

Preconditions:

Non-Standards Track Work Product

- A BPI test system is operational.
- A set of test transactions has been created.

Configure and initialize the Storage capability of the BPI Processing Layer.

Test Steps:

1. Submit a series of transactions to the BPI system.
2. Verify that the VSM generates proofs for each transaction.
3. Retrieve the stored proofs, state objects, associated data, and their histories from the Storage capability.
4. Validate that the retrieved information matches the original data and corresponds to the respective transactions.

Passing Criteria:

- Multiple transactions are successfully submitted to and performed on the BPI system.
- The VSM successfully generates proofs for each transaction.
- The VSM stores the generated proofs, state objects, associated data, and their histories in the Storage capability.
- The stored proofs, state objects, associated data, and their histories can be retrieved from the Storage capability.
- The retrieved information matches the original data and corresponds to the respective transactions.

[R271]

The integrity of proofs, transactions, state objects, and their data and histories MUST be cryptographically verifiable by the owners of the accounts associated with the proofs, transactions, state objects, and their data and history.

[R271] Testability:

Preconditions:

- A BPI test system with the necessary components, including accounts, proofs, transactions, state objects, and their data and histories is operational.
- A cryptographic verification mechanism with tooling to test integrity such as a Merkle Tree is properly configured and accessible to the account owners.
- One or more test transactions for the BPI.

Test Steps:

1. Generate a proof for a transaction on the BPI system.
2. Verify that the proof, along with the associated transaction, state object, and their data and histories, can be accessed by the respective account owners.
3. Using the chosen verification method, cryptographically verify the integrity of the proof, transaction, state object, and their data and histories using the available tooling for the chosen verification method.
4. Confirm that the account owners are able to successfully verify the integrity of the aforementioned components.

Passing Criteria:

- A proof is generated for a transaction.
- The proof, along with the associated transaction, state object, and their data and histories, can be accessed by the respective account owners.
- Using the chosen verification method and associated tooling, the account owners can successfully cryptographically verify the integrity of the proof, transaction, state object, and their data and histories.

[R272]

All updates to an agreement state and their associated accounts by a VSM MUST be communicated to all agreement counterparties through the Message capability in the BPI Middleware layer.

[R272] Testability:

Preconditions:

- A BPI test system with the necessary components, including agreement states, associated accounts, and the Message capability in the BPI Middleware layer.
- All counterparties to a test agreement are registered as BPI Subjects in the test BPI and connected to the BPI Message capability.
- There are one or more test transactions operating on the agreement state using a VSM.

Test Steps:

1. Submit one or more test transactions causing an update to an agreement state (state object) and its associated accounts using the VSM in the BPI system.
2. Verify that the generated state update message is processed by the Message capability.
3. Confirm that all agreement counterparties receive the state update message.
4. Validate that the state update message contains the relevant information about the update, such as the updated agreement state and the associated accounts.

Passing Criteria:

- A successful update is made to an agreement state and its associated accounts using the VSM.
- The update is successfully executed by the Message capability.
- The message contains the necessary information about the update.
- All agreement counterparties receive the message.

[R273]

A Proof-of-Correctness of a state transition and associated data required for proof verification generated by a VSM MUST be communicated to the CCSM Abstraction Layer for subsequent commitment to the CCSM utilized by the BPI through the Message capability in the BPI Middleware layer.

Note, see the BPI transaction lifecycle management flow in Fig. 10, in particular step (8).

[R273] Testability:

Preconditions:

- An operational BPI test system with the necessary components, including the VSM, CCSM Abstraction Layer, CCSM, and the Message capability in the BPI Middleware layer.
- One or more test transactions that trigger state transition.

Test Steps:

1. Submit a test transaction to the BPI
2. Verify that a Proof-of-Correctness for a state transition and its associated data using the VSM in the BPI system was successfully generated.
3. Verify that the generated Proof-of-Correctness was sent to the CCSM Abstraction Layer through the Message capability in the BPI Middleware layer.
4. Verify that the CCSM Abstraction Layer receives the Proof-of-Correctness message.
5. Validate that the Proof-of-Correctness message contains the necessary information for subsequent commitment to the CCSM.
6. Confirm that the CCSM Abstraction Layer successfully commits the Proof-of-Correctness to the CCSM.
7. Repeat Step 1. through 6. for all test transactions.

Passing Criteria:

Non-Standards Track Work Product

- For each test transaction and associated state transition, a Proof-of-Correctness is successfully generated using the VSM.
- The CCSM Abstraction Layer receives the Proof-of-Correctness message for each transaction.
- Each Proof-of-Correctness message contains the necessary information for subsequent commitment to the CCSM.
- The CCSM Abstraction Layer successfully commits each Proof-of-Correctness to the CCSM.

The following requirements are addressing the operating scenario where a BPI consists of more than one node. This is a perfectly feasible scenario with its pros and cons beyond the scope of this document to discuss. However, certain requirements need to be met for such a scenario to be operationally viable.

[O7]

A BPI MAY consist of more than one processing node.

This document will call such a structure a BPI network.

[O7] Testability:

Preconditions:

- An operational BPI test system with multiple processing nodes.
- One or more test transactions.

Test Steps:

1. Verify that the BPI system has been configured with multiple processing nodes.
2. Submit a transaction to the BPI system.
3. Confirm that the transaction is distributed and processed across multiple processing nodes.
4. Validate that at least one processing node executed the transaction.

Passing Criteria:

- Transactions are distributed and successfully processed across all the processing nodes.
- At least one processing node actively participates in the processing of transactions.
- The processing nodes effectively communicate and coordinate with each other during transaction processing.

[CR29]>[O7]

A BPI network executing and finalizing transactions MUST utilize a consensus algorithm fulfilling all requirements described in the Baseline CCSM Specification.

[CR29]>[O7] Testability:

Preconditions:

- An operational BPI network with multiple nodes.
- A configured consensus algorithm for the BPI network.
- One or more test transactions.

Test Steps:

1. Submit a transaction to the BPI network.
2. Monitor the network to ensure that the consensus algorithm is used for transaction finalization.
3. Repeat Step 1. and 2.
4. Validate that a consensus majority of nodes in the network agree on the order and validity of transactions.
5. Verify that the transactions are successfully executed and finalized based on the consensus algorithm.

Passing Criteria:

- The chosen consensus algorithm is utilized for transaction finalization.
- A consensus majority of nodes in the network agree on the order and validity of transactions.
- The submitted transactions are successfully executed and finalized based on the consensus algorithm.

[CR30]>[O7]

The consensus algorithm of a BPI network MUST have a time to consensus that is smaller than the time to consensus of the CCSM utilized by the BPI network.

[CR30]>[O7] Testability:

Preconditions:

- An operational BPI test network with multiple nodes.
- A configured consensus algorithm for the BPI network.
- One or more test transactions.

Test Steps:

1. Submit a test transaction to the BPI network.
2. Measure the time it takes for the BPI network to reach consensus on the transaction.
3. Measure the time it takes for the CCSM to reach consensus on the inclusion of the proof-of-correct-execution of the test transaction.
4. Compare the time to consensus of the BPI with the time to consensus of the CCSM.
5. Repeat Steps. 1. to 4. for all test transactions.

Passing Criteria:

- The BPI reaches consensus on the test transactions.
- The CCSM reaches consensus on the inclusion of the proof-of-correct-execution of the test transactions.
- The time to consensus of the BPI is smaller than the time to consensus of the CCSM.

[CR31]>[O7]

The consensus algorithm of a BPI network MUST have a time to finality that is smaller than the time to finality of the CCSM utilized by the BPI network.

[CR31]>[O7] Testability:

Preconditions:

- An operational BPI test network with multiple nodes.
- A configured consensus algorithm for the BPI network.
- One or more test transactions.

Non-Standards Track Work Product

Test Steps:

1. Submit a test transaction to the BPI network.
2. Measure the time it takes for the consensus algorithm of the BPI network to reach finality on the transaction.
3. Measure the time it takes for the consensus algorithm of the CCSM to reach finality on the inclusion of the proof-of-correct-execution of the test transaction.
4. Compare the time to finality of the BPI with the time to finality of the CCSM.
5. Repeat Steps. 1. to 4. for all test transactions.

Passing Criteria:

- The consensus algorithm of the BPI reaches finality on the test transactions.
- The consensus algorithm of the CCSM reaches finality on the inclusion of the proof-of-correct-execution of the test transactions.
- The time to finality of the BPI is smaller than the time to finality of the CCSM.

The above requirements are necessary such that transactions in the BPI cannot be altered after they have been committed to and finalized on the CCSM utilized by the BPI.

[CR32]>[O7]

A BPI network MUST reach consensus on both the order and the correct execution of transactions.

[CR32]>[O7] Testability:

Preconditions:

- An operational BPI test network with multiple nodes.
- A configured consensus algorithm for the BPI network for the order of transactions and the correct execution of transactions.
- One or more test transactions.

Test Steps:

1. Submit a set of transactions to the BPI network in a specific order.
2. Monitor the consensus process to ensure that all nodes participate in the consensus.
3. Verify that the BPI network reaches consensus on the order of the transactions based on the submission order.
4. Verify that the BPI network reaches consensus on the correct execution of the transactions.
5. Validate that all nodes in the BPI network have the same ordered and executed transactions.

Passing Criteria:

- The BPI network reaches consensus on the order of the transactions based on the submission order.
- The BPI network reaches consensus on the correct execution of the transactions.
- All nodes in the BPI network have the same ordered and executed transactions.

[CR33]>[O7]

A BPI network MUST use a common execution framework.

Note that if more than one execution framework were chosen, no consensus could be reached on the outcome of a transaction because the state representation is execution framework dependent e.g. Ethereum account state vs. a zero-knowledge-proof of account state.

[CR33]>[O7] Testability:

Preconditions:

- An operational BPI network with multiple nodes.
- One or more test transactions.

Test Steps:

1. Submit a set of transactions to the BPI network.
2. Monitor the execution of transactions across the network.
3. Validate that the transactions are executed on each node produce consistent and expected results.
4. Validate that the transactions that are executed on each node produce the same results on each node.

Passing Criteria:

- The transactions are executed on each node
- The transactions that are executed on each node produce the same result.

The last capability relevant for the transaction lifecycle management in the BPI processing layer that needs to be specified is the Storage capability.

The key discussion on the Storage capability is full data persistency (write many, read many) versus partial data persistency (write once, read many). An enterprise database such as MongoDB is an example of the former, and Ethereum is an example of the latter.

In the case of a BPI, this document needs to distinguish between proof, transaction, and state data and its history and metadata associated with state objects, etc. Given the need to maintain consistency between state data on the CCSM utilized by the BPI and state data in the BPI, the following requirement is important.

[R274]

Proof, transaction, and state object data and their histories together with their integrity proofs MUST be stored as partially persistent data in the BPI storage capability.

Partially persistent data is defined as append-only (write once, read many).

The Storage capability requirements are defined in section [7 General BPI Storage Capabilities](#).

[R274] Testability:

Preconditions:

- An operational BPI test system.
- Data structures and formats for proof, transaction, and state object data, including their histories and integrity proofs.
- A test set for a proof, and a state object, including its history and (CCSM) integrity proof.
- One or more test transactions and its expected results.

Test Steps:

1. Submit a test transaction.
2. Verify that the transaction is correctly executed and the resulting proof, state object, and history and integrity data are stored in the storage capability
3. Retrieve the stored data from the storage capability.

4. Validate that the retrieved data matches the expected data.
5. Attempt to modify the stored data.
6. Ensure that the modification is rejected, and the stored data remains unchanged.
7. Append additional data to the existing stored data by submitting another transaction.
8. Verify that all the data is successfully stored and retrievable.
9. Repeat Steps 1. through 8. for all test transactions.

Passing Criteria:

- The proof, transaction, and state object data are stored in the BPI storage capability without any modification.
- The retrieved stored data matches the originally submitted data.
- Modification of the stored data is rejected, and the stored data remains unchanged.
- Additional data can be successfully appended to the existing stored data using transactions.

7 General BPI Storage Capabilities

BPISTORAGEGENERAL

BPI storage is a key enabler to scale BPI stacks that are either data-intensive or data sensitive or both.

This document defines BPI data storage – outside of a CCSM – as the storing of information in a digital, machine-readable medium where the data stored is relevant for the proper functioning of the BPI stack.

The document defines storage capability requirements in the following areas:

- Security
- Privacy
- Integration
- Data Orchestration
- Partially Persistent Data
- Fully Persistent Data

7.1 BPI Storage Security

BPISTORAGESECURITY

As has been done throughout this document, there are BPI layer-specific security requirements for layers and/or components which are listed below. It is assumed that the BPI Storage capabilities are required to fulfill all BPI security requirements in [2 Design and Architecture](#).

[R275]

Data in transit in a BPI MUST be encrypted.

[R275] Testability:

Preconditions:

- There is a BPI test environment with at least two independent services processing data that communicate with each other
- The communication between the services is captured using a packet capture system
- The encryption algorithm(s) in the test meets the security requirements of this document, see [\[R12\]](#).

Test Steps:

1. Start the communication between the two services in the test environment using some plain text test data which is encrypted/decrypted by each service.
2. Capture the network traffic using the packet capture tool.
3. Inspect the captured packets to confirm that all data transmitted between the nodes is encrypted, and not in plain text, and can be decrypted using the key(s) used for encryption.
4. Repeat steps 1-3 for different types of data and communication scenarios (e.g., different data formats, different network topologies).

Expected Results: All data transmitted between the nodes is encrypted (does not appear in plain text).

The requirement is met, if all expected results are met. The requirement is not met, if any of the results are not meeting test expectations.

[R276]

Data at rest in a BPI MUST be encrypted.

[R276] Testability:

Preconditions:

- There is a BPI test environment that can store data
- The encryption algorithm(s) in the test meets the security requirements of this document, see [\[R12\]](#).

Test Steps:

1. Identify the location of the data at rest in the BPI.
2. Verify that the data is encrypted with the encryption/decryption key.
3. Attempt to decrypt the encrypted data without the encryption/decryption key and verify that it is not possible.
4. Repeat steps 1-3 for different types of data and storage scenarios (e.g., different data formats, different storage media).

Expected Results: The data at rest in the BPI is encrypted and cannot be decrypted without the encryption/decryption key.

The requirement is met, if all expected results are met. The requirement is not met, if any of the results are not meeting test expectations.

[R277]

BPI Storage arranged in a network MUST support pairwise key/identity relationships between storage nodes.

This is also known as a secure connection.

[R277] Testability:

Preconditions:

- There is a BPI test environment with at least two independent storage nodes that communicate with each other
- The communication between the services is configured to support pairwise key/identity relationships
- The encryption/decryption algorithm(s) in the test meets the security requirements of this document, see [\[R12\]](#).

Non-Standards Track Work Product

Test Steps:

1. Verify that each storage node has a unique identity and a corresponding public key.
2. Establish a pairwise key/identity relationship between two storage nodes.
3. Verify that the storage nodes can communicate securely using the established relationship.
4. Verify that the storage node does not provide access to the data stored except using the established relationship.
5. Repeat steps 2-4 for different pairs of storage nodes in the network.

Expected Results:

1. Each storage node has a unique identity and a corresponding public key.
2. The pairwise key/identity relationships between storage nodes are established.
3. The storage nodes can communicate securely using the established relationships.
4. It is not possible to access the data stored on one storage node from the other without using the established relationship.

The requirement is met, if all expected results are met. The requirement is not met, if any of the results are not meeting test expectations.

[R278]

BPI Storage MUST be compatible with commonly used external authentication services.

Non-normative examples of such authentication technologies are OAUTH [\[OAuth-2.0\]](#) , SAML [\[SAML\]](#) , OIDC [\[OIDC\]](#), AD/LDAP [\[ActiveDirectory\]](#).

[R278] Testability:

Preconditions:

- A BPI Storage e.g. the open-source [PostgreSQL](#) data base deployed in a BPI test environment with a BPI client and a BPI server
- An authentication service to be used with the BPI Storage such as [OAuth-2.0](#) between client and server.

Test steps:

1. Create a BPI Subject within the BPI with a public-private cryptographic key pair based on one of the cryptographic algorithms supported by the authentication service chosen for the BPI Subject
2. Create an authentication policy for the BPI Storage supported by each of the authentication services chosen for the test
3. Attempt to authenticate the BPI Subject with the BPI Storage using each of the identified external authentication services for the test.
4. Verify that the authentication of the BPI Subject is successful and the BPI Subject is granted the appropriate access to the BPI Storage as per the test authentication policy.
5. Repeat the above steps for each supported external authentication service.

Test pass criteria:

The test will pass if one or more of the external authentication services successfully authenticate a BPI Subject to the BPI Storage and the BPI Subject is granted the type of access to the BPI Storage as defined in the test policy.

Note that the test policy could be any combination of read and write access to the BPI Storage.

[R279]

BPI Storage MUST support roles & access management.

[R279] Testability:

Preconditions:

- Identify roles and levels of access required that need to be supported by the BPI Storage, based on the business requirements of the test use case.
- A BPI test system is up and running.
- Two or more BPI Subjects and BPI Subject Accounts created in the BPI

Test steps:

1. Create two or more test BPI Subjects and assign them to the identified roles and the level of access specified for each role.
2. Attempt to perform various operations on the BPI Storage using each test BPI Subject.
3. Verify that the operations are allowed or denied based on the access permissions associated with each of the BPI Subjects' role.
4. Attempt to perform operations that require different access permissions than the test BPI Subjects' role.
5. Verify that such operations are denied and an appropriate error message is displayed.
6. Repeat the above steps for each test role.

Test pass criteria:

- The test will pass if all test roles are successfully tested, and the test BPI Subjects are able to perform operations only as allowed by their role's permissions.
- Additionally, attempting operations that require different permissions than the test BPI Subjects' roles must be denied, and appropriate error messages must be displayed.

[R280]

BPI Storage MUST support policy management.

Policy management in the context of this document means the creation, reading of, updating and deletion or archiving of policies used in a BPI Storage. Examples of such policies to be managed are rules governing creation, modification, deletion, and retention of data.

[R280] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- The BPI storage component is designed to support policy management.
- A set of test policies has been configured in the BPI storage component.
- A BPI Subject has appropriate permissions for the test policies.

Test Steps:

1. Create a set of policies that the BPI storage component should be able to manage.
2. Using the BPI's user interface or API, attempt to create each of the policies defined in step 1.
3. Verify that each policy was created successfully and is stored in the BPI storage component.
4. Modify one or more of the policies created in step 2 and verify that the changes are applied correctly.
5. Attempt to delete one or more of the policies created in step 2 and verify that they are removed from the BPI storage component.
6. Attempt to retrieve a policy from the BPI storage component using the BPI's user interface or API and verify that the correct policy is returned.
7. Verify that an action on the BPI Storage component taken by the test BPI Subject is either allowed or denied by the set of policies created in step 2 and verify that the appropriate actions towards the test BPI Subject are taken based on the result of the policy evaluation of the test BPI Subjects attempted action.

Non-Standards Track Work Product

8. Repeat steps 2-7 for each type of policy created in step 1.
9. Verify that the BPI's user interface or API provides a clear way to manage policies.

Test Passing Criteria: The test will pass if all of the following criteria are met:

- All policies created in step 2 are successfully created, modified, and deleted, and are stored in the BPI storage component.
- Retrieval of policies in step 6 returns the correct policy for each policy created in step 2.
- Enforcement of policies in step 7 results in the appropriate actions for each policy created in step 2.
- The BPI's user interface or API provides an intuitive way to manage policies.
- There are no errors or exceptions during the test.

[R281]

BPI Storage MUST support Single-Sign-On (SSO).

See [\[SSO\]](#) also for the recommendations of the National Institute of Standards and Technology (NIST Guide to Secure Web Services).

[R281] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- The BPI storage component is designed to support SSO using a SSO provider.
- A BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. Attempt to access the BPI's user interface or API without providing any authentication credentials.
2. Verify that the BPI redirects the BPI Subject to a SSO login page or displays a message indicating that authentication is required.
3. Log in using a SSO provider that has been configured for the BPI.
4. Verify that the BPI grants access to the BPI Subject after successful authentication.
5. Attempt to access a protected resource or perform a protected action within the BPI.
6. Verify that the BPI grants access to the BPI Subject if the BPI Subject has appropriate permissions.
7. Attempt to access a protected resource or perform a protected action within the BPI using an invalid or expired SSO token.
8. Verify that the BPI denies access to the BPI Subject and provides an appropriate error message.
9. Attempt to log out of the BPI.
10. Verify that the BPI logs out the BPI Subject from the SSO provider and revokes the BPI Subject's BPI session.

Test Passing Criteria: The test will pass if

- The BPI redirects the BPI Subject to a SSO login page or displays a message indicating that authentication is required.
- Successful login using a SSO provider grants access to the BPI.
- Access to protected resources or actions within the BPI is granted only to BPI Subjects with appropriate permissions.
- Invalid or expired SSO tokens result in access being denied and an appropriate error message being displayed.
- Logging out of the BPI logs the BPI Subject out of the SSO provider and revokes the BPI Subject's BPI session.

[R282]

BPI Storage MUST support multi-factor authentication (MFA)

This document adopts [NIST's definition of MFA](#).

[R282] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- The BPI storage component is designed to support MFA.
- A BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. Attempt to access the BPI's user interface or API without providing any authentication credentials.
2. Verify that the BPI redirects the BPI Subject to a login page or displays a message indicating that authentication is required.
3. BPI Subject provides valid credentials to log in to the BPI.
4. Verify that the BPI grants access to the BPI Subject after successful authentication of the provided credentials.
5. Attempt to access a protected resource or perform a protected action within the BPI.
6. Verify that the BPI grants access to the BPI Subject if the BPI Subject has appropriate permissions.
7. Attempt to enable MFA for the BPI Subject account used to log in to the BPI.
8. Verify that the BPI supports multiple factors for MFA, such as SMS, email, or an authenticator app.
9. Attempt to log out and log back in to the BPI using only the credentials used before MFA was activated.
10. Verify that the BPI requires the BPI Subject to provide an additional factor of authentication for MFA after the initial login.
11. Attempt to access a protected resource or perform a protected action within the BPI.
12. Verify that the BPI grants access to the BPI Subject if the BPI Subject has appropriate permissions and has provided the required additional factor of authentication for MFA.
13. Attempt to disable MFA for the BPI Subject account used to log in to the BPI.
14. Verify that the BPI disables MFA for the BPI Subject account and allows the BPI Subject to log in with only the initial set of credentials.

Test Passing Criteria: The test will pass if

- The BPI redirects the BPI Subject to a login page or displays a message indicating that authentication is required.
- Successful login using the initial BPI Subject credentials grants access to the BPI.
- Access to protected resources or actions within the BPI is granted only to a BPI Subject with appropriate permissions.
- The BPI supports multiple factors for MFA, such as SMS, email, or an authenticator app.
- After enabling MFA, the BPI requires the BPI Subject to provide an additional factor of authentication for MFA after the initial login.
- Access to protected resources or actions within the BPI is granted only if the BPI Subject has provided the required additional factor of authentication for MFA.
- After disabling MFA, the BPI allows the BPI Subject to log in with only the initial credentials used by the BPI Subject.

[R283]

BPI Storage MUST support hardware security modules (HSM).

This document adopts the [NIST definition](#) and for further information, refer to [\[HSM\]](#).

[R283] Testability:

Preconditions:

Non-Standards Track Work Product

- A BPI test environment is installed and configured properly.
- The BPI storage component is configured to support an HSM.
- The HSM is installed and properly configured to work with the BPI storage component.
- A BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. Attempt to access the BPI's user interface or API without providing any authentication credentials.
2. Verify that the BPI redirects the BPI Subject to a login page or displays a message indicating that authentication is required.
3. Log in to the BPI.
4. Verify that the BPI grants access to the BPI Subject after successful authentication.
5. Verify that the BPI communicates with the HSM to perform the cryptographic operation.
6. Attempt to generate a cryptographic key using the HSM.
7. Verify that the BPI communicates with the HSM to generate the key.
8. Attempt to store a cryptographic key in the HSM.
9. Verify that the BPI communicates with the HSM to store the key securely.
10. Attempt to retrieve a cryptographic key from the HSM.
11. Verify that the BPI communicates with the HSM to retrieve the key securely.
12. Attempt to delete a cryptographic key from the HSM.
13. Verify that the BPI communicates with the HSM to delete the key securely.
14. Attempt to perform a cryptographic processing operation available in the chosen test HSM using a cryptographic key created during the test in the HSM.
15. Verify that the cryptographic processing operation produced a cryptographic valid result.

Test Passing Criteria: The test will pass if

- The BPI redirects the BPI Subject to a login page or displays a message indicating that authentication is required.
- Successful login grants access to the BPI.
- The BPI can generate a cryptographic key using the HSM.
- The BPI can store a cryptographic key securely in the HSM.
- The BPI can retrieve a cryptographic key securely from the HSM.
- The BPI can delete a cryptographic key securely from the HSM.
- The BPI can perform cryptographic processing operations available in the HSM.

7.2. BPI Storage Privacy

BPISTORAGEPRIVACY

As has been done throughout this document, there are BPI layer-specific privacy requirements for layers and/or components which are listed below. It is assumed that the BPI Storage capabilities are required to fulfill all BPI privacy requirements in section [2 Design and Architecture](#).

[R284]

Personal Identifiable Information (PII) MUST NOT be stored in a BPI.

[R284] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- The BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. Create a record that contains PII, such as a name, address, social security number, or phone number.
2. Attempt to store the record in the BPI using a write operation method, such as POST or PUT.
3. Verify that the BPI rejects the record and returns an error message indicating that the record contains PII and cannot be stored.
4. Create a record that does not contain PII, such as a random string or number.
5. Attempt to store the record in the BPI using a write operation method, such as POST or PUT.
6. Verify that the BPI accepts the record and stores it successfully.

Example Methods to identify PII:

- Use regular expressions or data pattern matching to identify common PII data elements in a record.
- Compare record data with a pre-defined list of PII data elements to identify PII.

Test Passing Criteria: The test will pass if

- The BPI rejects the storage of records that contain Personal Identifiable Information (PII) and returns an error message indicating that PII cannot be stored.
- The BPI accepts the storage of records that do not contain PII and stores them successfully.
- The BPI is able to identify PII using the specified methods.
- The BPI correctly identifies records that contain PII and rejects their storage.

Note: The test above only verifies write operations, as the requirement is related to the storage of PII. However, read operations should also be checked to ensure that PII cannot be accessed or returned in query results.

[D39]

BPI Storage arranged in a network SHOULD utilize privacy-preserving P2P message protocols.

[D39] Testability:

Preconditions:

- BPI Storage is arranged in a two node network that can support pairwise key/identity relationships between storage nodes in a testable manner (see [R278](#) for testing details)

Test Steps:

1. Using a pairwise key/identity relationship between the two BPI storage nodes, asymmetrically encrypt a message such as a data replication message.
2. Perform Test from [R275](#)

Test Passing Criteria: The test will pass if the test passing criteria in [R275](#) Testability are met.

7.3. BPI Data Orchestration

BPIDATAORCHESTRATION

Data Orchestration is an automated process for taking siloed data from multiple storage locations, combining and organizing it, and making it available for analysis.

Non-Standards Track Work Product

To accommodate a high-volume, Low Latency environment with many data changes, BPI Data Orchestration has the following requirements:

[R285]

Data Orchestration utilized in a BPI MUST NOT be a single point of failure.

[R285] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- The Data Orchestration component is designed to handle multiple nodes and is not a single point of failure.
- The BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. Verify that the Data Orchestration component consists of multiple nodes distributed across different servers or locations.
2. Simulate the failure of one of the Data Orchestration nodes by shutting down the node or disconnecting it from the network.
3. Attempt to perform a write operation on the BPI, such as POST or PUT.
4. Verify that the BPI can still perform the operation successfully without any errors or data loss.
5. Restore the failed Data Orchestration node to the network.
6. Verify that the restored node can synchronize with the other nodes in the Data Orchestration component and that the BPI continues to function correctly.

Test Passing Criteria: The test will pass if

- The Data Orchestration component used in the BPI consists of multiple nodes distributed across different servers or locations.
- The BPI is able to handle the failure of one or more Data Orchestration nodes without any errors or data loss.
- The restored Data Orchestration node is able to synchronize with the other nodes in the component and the BPI continues to function correctly.

Note: Additional tests may be required to verify that the Data Orchestration component can handle high traffic and is able to scale horizontally to accommodate additional nodes as needed.

[R286]

Data Orchestration utilized in a BPI MUST preserve data consistency from source to target within the BPI.

[R286] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- The data source is not directly connected to the data target within the BPI.
- The BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. Perform a write operation by the BPI Subject on the BPI, such as POST or PUT, with a specified set of data.
2. Verify that the data is successfully written to the BPI.
3. Perform a read operation on the BPI to retrieve the data that was written.
4. Check that the data retrieved matches the data that was written to the BPI.
5. Perform a read operation on the BPI data target system, the BPI Storage, to retrieve the data that was written to the BPI.
6. Check that the data retrieved from the BPI data target system matches the data that was written to the BPI.
7. Repeat steps 1-6 with multiple concurrent write operations.
8. Verify that all of the written data is consistent between the BPI target system.

Test Passing Criteria: The test will pass if,

- The written data is successfully written to the BPI.
- The data retrieved from the BPI matches the data that was written.
- The data retrieved from the BPI target system matches the data that was written to the BPI.
- All of the written data is consistent across both the BPI and the target system, even under multiple concurrent write operations.

To avoid subscribers seeing partial and/or inconsistent data, BPI Data Orchestration has the following requirements:

[R287]

Data Orchestration utilized in a BPI MUST implement transaction boundaries.

This means that a single BPI Subject's action can trigger atomic updates. A transaction boundary is defined as where a transaction begins or ends, where within the transaction all writes to a system are atomic, in that they either all complete, or are all reverted if any single write in a given transaction fails. An atomic update is defined as an indivisible and irreducible series of system operations such that either all occurs, or nothing occurs. An example of a transaction boundary is a "Create Invoice" transaction that creates an invoice in a system or fails if an error occurs.

[R287] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- Several transaction types are defined within the BPI as part of Data Orchestration, including what constitutes a correctly and incorrectly formed transaction, and how often a transaction of a specific transaction type should be made available to worksteps processing this transaction type.
- The BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. The BPI Subject submits a correctly formed transaction of a specific type on the BPI by making a POST request.
2. Verify that BPI Data Orchestration writes this transaction to one designated data orchestration entry point associated with the chosen transaction type.
3. Verify that BPI Data Orchestration writes the transaction from the designated entry point for the transaction type to all BPI Data Orchestration exit points associated with the chosen transaction type.
4. The BPI Subject submits an incorrectly formed transaction of a specific type on the BPI by making a POST request.
5. Verify that BPI Data Orchestration writes this transaction to one designated data orchestration entry point associated with the chosen transaction type.
6. Verify that BPI Data Orchestration Does not write the transaction from the designated entry point for the transaction type to any BPI Data Orchestration exit points associated with the chosen transaction type, and generates an error message relayed back to the BPI Subject.
7. Repeat steps 1-6 with multiple concurrent transactions of different types.

Test Passing Criteria: The test will pass if,

- All writes of the correctly and incorrectly formed transactions to the transaction type specific Data Orchestration entry points succeed.
- All writes of the correctly formed transactions to the transaction type specific Data Orchestration exit points succeed.
- All writes of the incorrectly formed transactions to the transaction type specific Data Orchestration exit points fail, and generate an error message relayed back to the BPI Subject.

Non-Standards Track Work Product

[R288]

Data Orchestration utilized in a BPI MUST commit to the exact order in which transactions are received by Data Orchestration.

[R288] Testability:

Preconditions:

- A BPI test environment is installed and configured properly.
- The Data Orchestration module is properly configured and enabled.
- The BPI Subject has appropriate permissions to access the BPI.

Test Steps:

1. The BPI Subject sends two or more transactions to the BPI in a specific order targeting a specific workstep with BPI State Object.
2. Verify that the transactions are received by the Data Orchestration module in the exact same order in which they were sent at a Data Orchestration entry point.
3. Verify that the transactions are committed by the Data Orchestration module in the exact same order in which they were received to one or more Data Orchestration exit points.
4. Verify that the committed transactions have been properly applied by the workstep to the BPI State Object and that the final state of the BPI State Object is consistent with the transactions that were sent.

Passing Criteria: The test will pass if all the following criteria are met,

- The transactions are received by the Data Orchestration module in the exact same order in which they were sent.
- The transactions are committed by the Data Orchestration module in the exact same order in which they were received.
- The committed transactions have been properly applied to the targeted BPI State Object and the final state of the BPI State Object is consistent with the transactions that were sent.

[R289]

Data Orchestration utilized in a BPI MUST support a consistent Data Orchestration state.

This can be achieved for example using a two-phase-lock commitment that ensures that a message log in the data orchestration module is idempotent, in other words is append only once per message and not more.

[R289] Testability:

Preconditions:

- A BPI test environment is deployed and running.
- The Data Orchestration module is properly configured and enabled.

Test Steps:

1. Send a BPI transaction request to the BPI though a POST or PUT request
2. Verify that the message log in the Data Orchestration module is updated only once for the transaction.
3. Repeat step 1 and 2 for multiple concurrent transactions.
4. Verify that the message log in the Data Orchestration module is updated only once for each of the concurrent transactions.
5. Introduce errors in the Data Orchestration module, such as network failures, and retry the transaction requests.
6. Verify that the message log in the Data Orchestration module is updated only once for each of the retried transactions.

Test Passing Criteria: The test passes if:

- The message log in the Data Orchestration module is updated only once for each transaction and for each concurrent transaction.
- The message log in the Data Orchestration module is not updated for failed transactions.
- The message log in the Data Orchestration module is updated only once for each retried transaction.

[R290]

Data Orchestration utilized in a BPI MUST support user-space processing.

In the context of this document, this requirement establishes a capability that allows for one or more computations outside a database to be triggered by a data change in the data storage system.

[R290] Testability:

Preconditions:

- A BPI test instance is set up and running.
- User-space processing functionality is enabled and configured such that a data change in a source system triggers at least one data change in a target system as defined in the user-space processing function.
- Make sure there is only one user-space processing function for the test.

Test Steps:

1. Add a new record to a BPI data storage system as the source system defined in the user-space processing function.
2. Verify that the user-space processing is triggered by the data change.
3. Check that the user-space processing function is executed correctly and one or more data changes are made in the defined target system as per the functions definition.
4. Make changes to the newly created record.
5. Verify that the user-space processing is triggered again by the data change.
6. Check that the user-space processing function is executed correctly again as per the functions definition.
7. Add a new record to another BPI data storage system not defined as the source system in the user-space processing function.
8. Verify that the user-space processing is not triggered by the data change.

Test Passing Criteria:

- The user-space processing is triggered by the data change in a BPI storage system.
- The user-space processing function is executed correctly for both steps 3 and 6.
- The processing function is not triggered for any data change that is not specified in the configuration.
- The BPI storage system remains stable and consistent throughout the test.

[R291]

Data Orchestration utilized in a BPI MUST NOT make assumptions about the uptime of a Data Orchestration consumer.

[R291] Testability:

Preconditions:

- BPI test environment is set up and running.
- Data Orchestration is implemented and functional.

Non-Standards Track Work Product

- Data Orchestration consumer is registered.

Test Steps:

1. Simulate a scenario where the Data Orchestration consumer experiences an unexpected downtime.
2. Verify that the Data Orchestration continues to operate normally without any impact to the BPI's functionality.
3. Bring the Data Orchestration consumer back online.
4. Verify that the Data Orchestration resumes sending data to the consumer without any data loss or corruption.

Test Passing Criteria:

- Step 2 must complete without any errors or issues.
- Step 4 must complete without any data loss or corruption.
- The BPI must remain operational and functional even when the Data Orchestration consumer is down.

[R292]

Data Orchestration utilized in a BPI MUST isolate Data Orchestration data sources from Data Orchestration consumers to which data from the Data Orchestration data sources is delivered via the BPI's Data Orchestration component.

[R292] Testability:

Preconditions:

- A BPI test system with Data Orchestration component is set up and running.
- Data sources and data source associated data consumers have been identified and configured in the BPI system and its Data Orchestration component.

Test Steps:

1. Send data from a data source to the Data Orchestration component of the BPI system.
2. Verify that the data is received and stored by the Data Orchestration component.
3. Check that the data source is not accessible to any Data Orchestration consumers.
4. Verify that the Data Orchestration component delivers the received data to the consumer.
5. Check that the Data Orchestration consumer does not have access to the data source that provided the data.
6. Repeat steps 1-5 for multiple data sources and consumers.

Test Passing Criteria:

- The Data Orchestration component should successfully receive and store data from the data source.
- The data source should not be accessible to any Data Orchestration consumers.
- The Data Orchestration component should successfully deliver data to the consumer associated with the data source.
- The Data Orchestration consumer should not have access to * the data source that provided the delivered data.
- The test should pass for all tested data sources and consumers.

[R293]

Data Orchestration utilized in a BPI MUST support Low Latency.

Low latency in this context refers to a latency that does not impact the overall system latency of the BPI.

[R293] Testability:

Preconditions:

- A BPI test system with Data Orchestration capability is set up and running.
- A test data source and consumer are set up within the BPI.
- The BPI has a defined threshold for "Low Latency".

Test Steps:

- Send a request to the test data source to initiate a data change.
- Measure the time it takes for the data change to propagate through the Data Orchestration component and reach the test consumer.
- Compare the measured time with the defined "Low Latency" threshold.

If the measured time is below the defined threshold, the test passes. Otherwise, the test fails.

Test Passing Criteria:

- The test is considered passed if the measured time for the data change to propagate through the Data Orchestration component and reach the test consumer is below the defined "Low Latency" threshold.

[R294]

Data Orchestration utilized in a BPI MUST be scalable and highly available such that overall system latency is not impacted when volume meaningfully and rapidly changes at any point in time.

[R294] Testability:

Preconditions:

- A BPI test system is set up and running with Data Orchestration implemented.
- There are enough resources available to simulate varying volume loads on the system.
- Define minimum and maximum processing volume for the test.
- Define acceptable Data Orchestration latencies for the test.
- Define BPI system latencies for the test.

Test Steps:

1. Increase the volume of data being processed by the Data Orchestration component to a significant level.
2. Verify that the overall system latency is within acceptable limits.
3. Gradually increase the volume of data being processed until it reaches a peak level.
4. Verify that the overall system latency is still within acceptable limits.
5. Reduce the volume of data being processed back to the initial level.
6. Verify that the overall system latency returns to its previous level.

Passing Criteria:

- In steps 2 and 4, the overall system latency should not exceed the acceptable limit specified for the BPI.
- In step 6, the overall system latency should return to its previous level.

Non-Standards Track Work Product

- The test should be repeated multiple times with varying volume loads, and the passing criteria should be met consistently.

The BPI Data Orchestration must include the following four components:

[R295]

Data Orchestration utilized in a BPI MUST include a fetcher capability that extracts changes from the data source or another bus component.

[R295] Testability:

Preconditions:

- A BPI test instance with a data source and a Data Orchestration component.

Test Steps:

1. Inject a test record into the data source.
2. Wait for a reasonable period for the fetcher capability of the Data Orchestration component to extract the changes from the data source.
3. Retrieve the extracted record from the Data Orchestration component.
4. Verify that the retrieved record matches the injected record in the data source.

Test Passing Criteria:

- The injected record in the data source and the retrieved record from the Data Orchestration component must match.
- The fetcher capability must extract the changes from the data source in a reasonable period of time.
- The extracted changes must be available for retrieval from the Data Orchestration component.

Note: The test steps can be modified to include multiple records and various types of data sources to test the scalability and flexibility of the fetcher capability.

[R296]

Data Orchestration utilized in a BPI MUST include a log store that caches the generated data change stream in a BPI.

[R296] Testability:

Preconditions:

- A BPI test system is set up and running with Data Orchestration enabled.
- A log store is configured and connected to the Data Orchestration component.

Test Steps:

1. Make a change to the data source connected to the BPI.
2. Verify that the change is captured in the log store of the Data Orchestration component.
3. Make another change to the data source.
4. Verify that the second change is also captured in the log store of the Data Orchestration component.
5. Verify that the order of changes in the log store matches the order in which they were made to the data source.
6. Disable the log store and make a change to the data source.
7. Verify that the change is not captured in the log store.

Test Passing Criteria:

- The changes made to the data source are captured in the log store.
- The order of changes in the log store matches the order in which they were made to the data source.
- The log store can be enabled or disabled as desired, and only captures changes when enabled.

[R297]

Data Orchestration utilized in a BPI MUST include a snapshot store that stores a moving snapshot of the generated change data stream.

[R297] Testability:

Preconditions:

- The Data Orchestration layer has been properly installed and configured in the BPI test environment.
- A data source is connected and sending data to the Data Orchestration layer.

Test Steps:

1. Send a set of test data to the Data Orchestration layer.
2. Verify that the data is properly received and stored in the log store.
3. Verify that the snapshot store has captured the data as a moving snapshot from the log store.
4. Change the data source by updating the data.
5. Verify that the updated data is properly received and stored in the log store.
6. Verify that the snapshot store has captured the updated data as a moving snapshot from the log store.
7. Repeat steps 4-6 with different sets of data to ensure that the snapshot store can capture a moving snapshot of any generated change data stream.

Test Passing Criteria:

- The test passes if all test steps have been successfully executed without errors or failures.
- The snapshot store should be able to store a moving snapshot of the generated change data stream.
- The snapshot store should capture any changes to the data source in the moving snapshot from the log store.

[R298]

Data Orchestration utilized in a BPI MUST include a subscription client pulling change events across the Data Orchestration component and delivering them to a service in a BPI with Low Latency.

[R298] Testability:

Preconditions:

- A BPI test system is set up and running
- The Data Orchestration component is configured and operational
- The service that will receive change events is set up and running
- Define Low Latency tolerance values for the Data Orchestration component relative to overall BPI system latency targets.

Test Steps:

Non-Standards Track Work Product

1. Generate data changes in the data source used by the Data Orchestration component
2. Verify that the subscription client is pulling change events from the Data Orchestration component
3. Verify that the change events are being delivered to the service
4. Increase the volume and frequency of data changes in the data source and repeat steps 2-3
5. Verify that the overall system latency of the BPI is not impacted by the increase in data changes

Test Passing Criteria:

- Step 1 is successful
- Steps 2-3 are successful and the change events are being delivered to the service with Low Latency
- Step 4 is successful and the change events are being delivered to the service with Low Latency
- Step 5 is successful and the overall system latency of the BPI is not impacted by the increase in data changes

7.4 BPI-External Storage: Edge Storage

BPIEDGE STORAGE

There are operating scenarios where it could be necessary that BPI data is replicated outside of a BPI such as to avoid having to rebase the state of a system or record due to an accidental data update if the correct state is not readily accessible to enforce system-of-record access policies.

[\[R299\]](#)

BPI Edge Storage MUST ensure eventual consistency between edge storage and BPI under a weak synchrony assumption.

Weak synchrony in this context means:

- All messages will eventually reach their intended recipients
- After a certain, yet unknown, time the network will become synchronous again

[\[R299\]](#) Testability:

Preconditions:

- A test BPI Edge Storage is set up and configured
- A BPI test system is running and accessible
- A set of data is stored in the BPI Edge Storage
- The network between BPI and BPI Edge Storage is delivering messages in finite time as set by the test.

Test Steps:

1. Store a set of data in BPI Edge Storage.
2. Verify that the data is available in BPI Edge Storage and not in the BPI.
3. Introduce a delay in the network that lasts longer than the known average duration to send data from the BPI Edge Storage to the BPI.
4. Update the data in the BPI Edge Storage.
5. Verify that the data update is eventually propagated to the BPI and the data in BPI is consistent with the updated data in BPI Edge Storage.
6. Remove the network delay and verify that the updated data is available in BPI with minimal delay.

Test Passing Criteria:

- The data stored in BPI Edge Storage is available and consistent with BPI under normal network conditions.
- The data update in BPI Edge Storage is eventually propagated to BPI and the data in BPI is consistent with the updated data in BPI Edge Storage.
- The delay introduced in the network does not result in data loss or corruption.
- The updated data is available in BPI with minimal delay after the network delay is removed.

[\[R300\]](#)

Data replication conflicts in BPI Edge Storage MUST be automatically detectable.

[\[R300\]](#) Testability:

Preconditions:

- A BPI Edge Storage system is properly set up and configured.
- There are at least two storage nodes that are replicating data.
- The data being replicated has the potential to cause conflicts.

Test Steps:

1. Write a piece of data to BPI Edge Storage.
2. Simulate a network partition between the storage nodes, so that each node continues to receive writes while unable to communicate with each other.
3. Update the same piece of data on both storage nodes while the network partition is in place.
4. Rejoin the network partition.
5. Verify that the BPI Edge Storage system has detected a conflict.
6. Verify that the BPI Edge Storage system has resolved the conflict by using a conflict resolution strategy, such as last write wins or a custom resolution function.

Test Passing Criteria:

- The BPI Edge Storage system detects the conflict within a reasonable amount of time, such as a few seconds.
- The conflict is resolved automatically by the BPI Edge Storage system.
- The resolution strategy used by the BPI Edge Storage system aligns with the chosen conflict resolution approach.
- The resolution is propagated to all nodes in the storage cluster, so that all nodes eventually have the same data.

[\[R301\]](#)

Data replication conflicts in BPI Edge Storage MUST be resolvable either automatically or manually.

[\[R301\]](#) Testability:

Preconditions:

- BPI Edge Storage is set up and running.
- There are multiple nodes for data replication in BPI Edge Storage.
- The BPI Edge Storage has passed the test for [\[R300\]](#).

Test Steps:

1. Simulate a data replication conflict scenario by creating two or more replicas of the same data item with different values in different nodes of BPI Edge Storage that cannot be automatically resolved by the BPI Edge Storage Data Conflict Resolution capability.

Non-Standards Track Work Product

2. Check whether BPI Edge Storage can automatically detect and resolve the data replication conflict. If yes, then repeat Step 1.
3. If BPI Edge Storage does not automatically resolve the conflict, a human-readable message must be created and a manual attempt to resolve the conflict must be possible through a user interface.
4. Verify that the data replication conflict has been successfully resolved through the user interface and only one replica of the data item with the correct value remains.

Test Passing Criteria:

- The test passes if BPI Edge Storage can automatically detect the data replication conflict and cannot resolve it automatically.
- The manual conflict resolution results in only one replica of the data item with the correct value remaining in BPI Edge Storage.
- The time taken to detect and resolve the conflict is within acceptable limits.

[R302]

BPI Edge Storage MUST use a secure and privacy-preserving wire protocol for communication.

[R302] Testability:

Preconditions:

- A BPI test system is up and running.
- BPI Edge Storage is installed and configured.
- BPI Edge Storage is using a wire protocol for communication.

Test steps:

1. Verify that a wire protocol is implemented in BPI Edge Storage.
2. Attempt to intercept communication between BPI Edge Storage and the BPI.
3. Verify that intercepted communication cannot be deciphered by analyzing the data from the wire protocol.
4. Verify that sensitive data in the communication is encrypted using the identified encryption algorithm in the wire protocol.
5. Verify that the protocol is resistant to person-in-the-middle attacks.

Test Passing criteria:

- The identified protocol is implemented in BPI Edge Storage.
- Intercepted communication cannot be deciphered by analyzing the data from the wire protocol.
- Sensitive data in the communication is encrypted using the identified protocol.
- The wire protocol is resistant to person-in-the-middle attacks.

[D40]

BPI Edge Storage SHOULD be able to cryptographically sign messages.

[D40] Testability:

Preconditions:

- A BPI Edge Storage is installed and configured properly.
- Cryptographic keys for message signing are generated and available.

Test Steps:

1. Send a message to the BPI Edge Storage.
2. Verify that the message received by BPI Edge Storage can be cryptographically signed.
3. Verify that BPI Edge Storage is able to sign the message with the cryptographic keys.
4. Send the signed message from BPI Edge Storage back to the message origin.
5. Verify the message origin has received the signed message from the BPI Edge Storage
6. Verify that the signature of the message sent by the BPI Edge Storage is valid using the corresponding cryptographic keys.

Test Passing Criteria:

- The message is successfully signed by BPI Edge Storage with the provided cryptographic keys.
- The signature of the message is verified as valid using the corresponding cryptographic keys.

[D41]

BPI Edge Storage SHOULD be discoverable by BPI Workgroup members or their delegates within a BPI.

[D41] Testability:

Preconditions:

- A BPI test system is up and running.
- A BPI Workgroup has been established with at least one member.
- BPI Edge Storage has been set up and is operational within the BPI.
- The BPI Workgroup members or their delegates have access to the BPI network and the necessary permissions to discover and access BPI Edge Storage.
- There exists one or more methods within the BPI to discover resources associated with or operated by the BPI, such as service discovery protocols or manual configuration.

Test steps:

1. Make BPI Edge Storage discoverable by BPI Workgroup members or their delegates within the BPI using the BPI discovery mechanism.
2. Attempt to discover BPI Edge Storage using various BPI discovery methods.
3. Verify that BPI Edge Storage has been successfully identified and can subsequently be accessed by BPI Workgroup members or their delegates.
4. Verify that BPI Edge Storage is accessible only to authorized parties and that unauthorized access attempts are rejected.
5. Attempt to repeat Steps 1. to 4. after the BPI Edge Storage has been removed from the BPI discovery mechanism

Test Passing criteria:

- BPI Edge Storage is discoverable by BPI Workgroup members or their delegates within the BPI.
- BPI Edge Storage can be accessed by BPI Workgroup members or their delegates.
- BPI Edge Storage is accessible only to authorized parties and unauthorized access attempts are rejected.
- The BPI Edge Storage is not discoverable in Step 5.

[R303]

BPI Edge Storage MUST support BPI identifiers and identity as defined in this document.

See section [3 Identifiers, Identity and Credential Management](#).

Non-Standards Track Work Product

[R303] Testability:

Preconditions:

- A BPI test system is up and running
- The BPI has assigned a BPI Identifier and Identity to the BPI Edge Storage
- The BPI Identifier, Identity, and its associated credentials for the BPI Edge Storage are stored in the BPI Edge Storage and in the BPI

Test Steps:

1. Verify that the BPI Edge Storage is able to accept and store the BPI Identifier and Identity.
2. Verify that the BPI Edge Storage is able to store and protect the associated credentials for the BPI Identifier and Identity.
3. Verify that the BPI Edge Storage is able to use the BPI Identifier and Identity to interact with other BPI 4. components, such as other BPI Storage.
4. Verify that the BPI Edge Storage is able to correctly identify itself to other BPI components using the assigned BPI Identifier and Identity.

Test Passing Criteria:

- The BPI Edge Storage is able to accept and store the BPI Identifier and Identity without error.
- The BPI Edge Storage is able to store and protect the associated credentials for the BPI Identifier and Identity without error.
- The BPI Edge Storage is able to use the BPI Identifier and Identity to interact with other BPI components without error.
- The BPI Edge Storage is able to correctly identify itself to other BPI components using the assigned BPI Identifier and Identity without error.

[R304]

BPI Edge Storage MUST support Partially Persistent Data and Fully Persistent Data.

See section [7.5 BPI-Internal Storage](#) requirements for security, privacy, and integration.

[R304] Testability:

Preconditions:

- A BPI test system is up and running.
- BPI Edge Storage is installed and configured with appropriate access credentials.

Test Steps:

1. Create a new Partially Persistent Data object in BPI Edge Storage.
2. Verify that the object can be accessed and updated without affecting newer versions.
3. Create a new Fully Persistent Data object in BPI Edge Storage.
4. Verify that all versions of the object can be accessed and updated independently.

Test Passing Criteria:

- The Partially Persistent Data object can be accessed and updated without affecting newer versions.
- The Fully Persistent Data object allows all versions to be accessed and updated independently.

7.5 BPI-Internal Storage

BPIINTERNALSTORAGE

There are two storage types BPI storage systems can utilize, fully or partially persistent storage.

[Fully Persistent Data storage](#) as one possible option for BPI storage can be characterized as Write many, Read many.

[Partially Persistent Data storage](#) as one possible option for BPI storage can be characterized as Write once, Read many.

There are two deployment options – centralized or distributed/decentralized deployment.

This document lists the requirements for either option and indicates the differences between partially and fully persistent data storage where required.

7.5.1 BPI Storage: Centralized Deployment

BPIINTERNALSTORAGECENTRALIZED

Since BPIs are typically used in an enterprise context, BPI Storage ideally has characteristics of commonly utilized enterprise-grade database solutions.

Characteristics of enterprise-grade database solutions are but not limited to:

- *Support for large number (> 1,000) of Parallel Queries*
- *Multi-process support where several processes can be handled by splitting workload between them*
- *Support for database clustering to process high data volumes in short periods of time (sub second processing)*
- *Security features that adhere to established industry security standards such as the US Federal Information Processing Standard [FIPS](#) or [ISO 27001](#)*

[O8]

Centralized BPI Storage MAY be partially persistent.

[O8] Testability:

Preconditions:

- A BPI test system is up and running.
- The Centralized BPI Storage has been set up and configured.

Test Steps:

1. Create a test data set and add it to the Centralized BPI Storage.
2. Modify the test data set and add it to the Centralized BPI Storage.
3. Verify that the latest version of the test data set is retrievable.
4. Verify that the previous version of the test data set is retrievable.
5. Modify the test data set again and add it to the Centralized BPI Storage.
6. Verify that the latest version of the test data set is retrievable.
7. Verify that the previous version of the test data set is still retrievable.
8. Repeat steps 5-7 with multiple data sets.
9. Verify that the Centralized BPI Storage is functioning properly and no data is lost or corrupted.

Test Passing Criteria:

Non-Standards Track Work Product

- All test steps are completed successfully.
- The latest and previous versions of the test data sets are retrievable from the Centralized BPI Storage.
- The Centralized BPI Storage is functioning properly and no data is lost or corrupted.

[CR34]>[O8]

Partially Persistent BPI Storage MUST be append-only.

Non-normative examples of such data bases are, but not limited to, OracleDB, MongoDB, Postgres, Cassandra, and DynamoDB.

[CR34]>[O8] Testability:

Preconditions:

- A BPI test system is up and running.
- A Partially Persistent BPI Storage has been set up.
- The system is capable of generating test data to append to the storage.

Test Steps:

1. Verify that the Partially Persistent BPI Storage is empty initially.
2. Append test data to the storage using the BPI API or other appropriate means.
3. Retrieve the appended data and verify its correctness.
4. Attempt to modify or delete the appended data.
5. Verify that modification or deletion of the appended data is not possible.

Test Passing Criteria: The test will pass if,

- Step 1 is successful and the storage is initially empty.
- Step 2 is successful and the test data is appended to the storage.
- Step 3 is successful and the retrieved data matches the test data.
- Step 4 is unsuccessful and the appended data remains unmodified and undeleted.
- Step 5 is successful and modification or deletion of the appended data is not possible.

7.5.2 BPI Storage: Decentralized Deployment

BPIINTERNALSTORAGEDECENTRALIZED

[R305]

BPI Storage MUST support authenticated naming systems.

An authenticated naming system in the context of this document is defined as a security protocol that enables a named entity such as an internet domain to be bound to cryptographic material such as a public key that allows for cryptographic authentication of the named entity. An example is a W3C DID or DNS-based Authentication of Named Entities [DANE].

Non-normative examples include, but are not limited to, certificate authorities or a self-certifying PKI namespace.

[R305] Testability:

Preconditions:

- A BPI test system is up and running.
- The BPI Storage has been configured with an authenticated naming system, such as W3C DID or DNS-based Authentication of Named Entities.

Test Steps:

1. Verify that the BPI Storage supports the authenticated naming system that has been configured.
2. Create a new record in the BPI Storage using the authenticated naming system.
3. Retrieve the record from the BPI Storage and verify that it contains the expected authenticated name and associated cryptographic material.
4. Attempt to retrieve the record using an incorrect authenticated name and verify that the retrieval fails.
5. Update the record in the BPI Storage with a new authenticated name and associated cryptographic material.
6. Retrieve the updated record from the BPI Storage and verify that it contains the new authenticated name and associated cryptographic material.

Test Passing Criteria:

- The BPI Storage supports the configured authenticated naming system.
- The created record contains the expected authenticated name and associated cryptographic material.
- The retrieval of the record using an incorrect authenticated name fails.
- The updated record contains the new authenticated name and associated cryptographic material.

[R306]

BPI Storage MUST support a data exchange protocol that allows for large blocks of data to be replicated.

A large data block in this document is defined to be larger than 1MB but less than 128MB.

[R306] Testability:

Preconditions:

- A BPI test system is up and running
- The BPI Storage is operational and has enough free space to store large data blocks
- The data exchange protocol used by the BPI Storage is configured to handle large data blocks of up to 128MB

Test steps:

1. Create a large data block of size 10MB.
2. Attempt to replicate the large data block to the BPI Storage.
3. Verify that the data block was successfully replicated to the BPI Storage and that its integrity was preserved during the replication process.
4. Create a second large data block of size 100MB.
5. Attempt to replicate the second large data block to the BPI Storage.
6. Verify that the data block was successfully replicated to the BPI Storage and that its integrity was preserved during the replication process.
7. Attempt to replicate a data block larger than 128MB to the BPI Storage.
8. Verify that the replication failed due to the size limit imposed by the data exchange protocol.

Test Passing criteria:

- The first large data block and the second large data block are successfully replicated to the BPI Storage and their integrity is preserved.
- The replication of a data block larger than 128MB to the BPI Storage fails.

Non-Standards Track Work Product

[R307]

BPI Storage MUST support a routing protocol that enables locating data peers and data objects.

Non-normative examples are [libp2p](#) or distributed hash tables [\[DHT\]](#).

[R307] Testability:

Preconditions:

- A BPI test system is up and running
- BPI Storage is configured and operational
- BPI Storage has at least two data peers with data objects available
- Routing protocol is enabled and properly configured in BPI Storage

Test Steps:

1. Verify that BPI Storage is able to discover and locate data peers through the routing protocol
2. Verify that BPI Storage is able to locate data objects in a specific data peer through the routing protocol
3. Verify that BPI Storage is able to route requests to the appropriate data peer based on the requested data object
4. Verify that BPI Storage is able to handle changes in data peer availability or data object locations and update the routing information accordingly
5. Verify that the routing protocol is able to handle a large number of data peers and data objects without significant degradation of performance

Test Passing Criteria:

- All test steps must pass without errors
- The routing protocol must be able to locate data peers and data objects with a success rate of at least 95%
- The routing protocol must be able to handle changes in data peer availability or data object locations within 30 seconds
- The routing protocol must not significantly degrade the performance of BPI Storage, with requests being handled within 500 milliseconds on average

[R308]

BPI Storage MUST support a Network Protocol that handles all of:

- NAT traversal such as hole punching, port mapping, and relay
- Multiple transport protocols
- Encryption, signing, or clear text communications
- Multi-multiplexes such as Multiplex connections, streams, protocols, peers

[R308] Testability:

Preconditions:

- A BPI test system is up and running
- BPI Storage is installed and running
- Network connectivity is available between BPI Storage and other network devices

Test Steps:

1. Test NAT traversal by verifying that BPI Storage can successfully perform hole punching, port mapping, and relay.
 - Connect two network devices on different NATs to the BPI Storage
 - Verify that the devices can establish a connection and communicate with each other through the BPI Storage using hole punching, port mapping, and relay techniques.
2. Test multiple transport protocols support
 - Connect a network device to the BPI Storage using TCP protocol
 - Verify that the device can communicate with BPI Storage using TCP protocol
 - Connect another network device to the BPI Storage using UDP protocol
 - Verify that the device can communicate with BPI Storage using UDP protocol
3. Test Encryption, signing, or clear communications support
 - Enable encryption and signing for communications between BPI Storage and a network device
 - Verify that communications between BPI Storage and the device are encrypted and signed
 - Disable encryption and signing for communications between BPI Storage and the device
 - Verify that communications between BPI Storage and the device are unencrypted
4. Test Multi-multiplexes support
 - Connect multiple network devices to the BPI Storage
 - Verify that the devices can establish multiple connections simultaneously through BPI Storage
 - Verify that the devices can communicate with each other through BPI Storage using the established connections.

Passing Criteria:

- NAT traversal, multiple transport protocols, encryption, signing, and clear communications, and multi-multiplexes tests should pass without any errors or failures.

[R309]

Fully Persistent BPI Storage MUST support Generalized Time Stamps.

Non-normative examples are conflict-free replicated data types [\[CRDT\]](#) or Interval Tree Clocks [\[ITC\]](#) to ensure eventual data consistency.

[R309] Testability:

Preconditions:

- A BPI test system is up and running.
- The BPI Storage system is fully persistent and supports Generalized Time Stamps.
- The BPI Storage system is configured to use an eventual data consistency condition and conflict-free data replication based on Generalized Time Stamps.
- There are at least two clients that can write to the BPI Storage independently.

Test Steps:

1. Write data to one replica of the Fully Persistent BPI Storage.
2. Read the data from all replicas of the Fully Persistent BPI Storage.
3. Verify that the data is consistent across all replicas and reflects the latest changes.
4. Modify the data in one replica of the Fully Persistent BPI Storage.
5. Write the modified data to one replica of the Fully Persistent BPI Storage.
6. Read the modified data from all replicas of the Fully Persistent BPI Storage.
7. Verify that the modified data is consistent across all replicas and reflects the latest changes.
8. Write data to two replicas of the Fully Persistent BPI Storage where the data written to the first replica is not consistent with the data written to the second replica and the first data set is written by client 1 and data set two is written by client 2, and there is a time delay between first and second data set.

Non-Standards Track Work Product

9. Verify that the chosen Generalized Time Stamp mechanism works properly by successfully repeating steps 2. through 7. where only the first data set is replicated.

Test Passing Criteria:

- The data written and modified in the Fully Persistent BPI Storage is consistent across all replicas.
- The data reflects the latest changes made to it.
The BPI Storage system has successfully stored data conflict-free using Generalized Time Stamps with eventual data consistency conditions, and the data can be retrieved without inconsistencies.
- The Fully Persistent BPI Storage provides full data persistence.

[O9]

Decentralized BPI Storage MAY be partially persistent.

[O9] Testability:

Preconditions:

- A BPI test system is up and running.
- The BPI decentralized storage has been properly set up and configured.
- The BPI supports both fully and partially persistent data storage.

Test Steps:

1. Create a new data object and store it in the decentralized BPI storage with partially persistent storage settings.
2. Retrieve the data object from the decentralized BPI storage.
3. Verify that the retrieved data object is the same as the one that was originally stored, with respect to its contents and associated metadata.
4. Append new data to the partially persistent data object in the decentralized BPI storage.
5. Retrieve the updated data object from the decentralized BPI storage.
6. Verify that the retrieved data object contains the newly appended data, with respect to its contents and associated metadata.
7. Store a new data object in the decentralized BPI storage with fully persistent storage settings.
8. Retrieve the fully persistent data object from the decentralized BPI storage.
9. Verify that the retrieved data object is the same as the one that was originally stored, with respect to its contents and associated metadata.
10. Attempt to append new data to the fully persistent data object in the decentralized BPI storage.
11. Verify that the attempt to append data fails, since fully persistent data objects cannot be appended.

Test Passing Criteria:

- Test steps 1-6 and 8-9 must pass, indicating that partially persistent data can be properly stored, retrieved, and updated in the decentralized BPI storage.
- Test steps 7 and 10-11 must pass, indicating that fully persistent data can be properly stored and retrieved, and that attempts to append data to fully persistent data objects fail as expected.
- The generalized time stamp testing should ensure eventual data consistency in all test steps.

[CR35]>[O9]

Partially Persistent BPI Storage MUST support Generalized Time Stamps or consensus protocols that guarantee eventual data consistency.

[CR35]>[O9] Testability:

Preconditions:

- A BPI test system is up and running
- The partially persistent BPI storage has been assigned a unique identifier and associated credentials
- The BPI storage has been configured to support Generalized Time Stamps or consensus protocols for eventual data consistency

Test Steps:

1. Write a data block of size 2MB to the partially persistent BPI storage using the supported Generalized Time Stamps or consensus protocols for eventual data consistency
2. Verify that the data block is successfully written to the storage with the correct timestamp or consensus protocol signature
3. Update the data block with a modification of size 500KB using the supported Generalized Time Stamps or consensus protocols for eventual data consistency
4. Verify that the modification is successfully applied to the data block and the updated timestamp or consensus protocol signature is correct
5. Delete the data block using the supported Generalized Time Stamps or consensus protocols for eventual data consistency
6. Verify that the data block is successfully deleted from the storage and the timestamp or consensus protocol signature is correct

Test Passing Criteria:

- All test steps are successfully executed without any errors
- The data block is successfully written to the partially persistent BPI storage with the correct timestamp or consensus protocol signature
- The modification to the data block is successfully applied and the updated timestamp or consensus protocol signature is correct
- The data block is successfully deleted from the storage and the timestamp or consensus protocol signature is correct

8 BPI External Data Inputs

bpiextdatainputs

This section of the document focuses on the requirements and considerations related to the input of external data into a BPI workstep. Specifically, it addresses:

- internal authoritative data - data sourced from a single authoritative source, internal to one of the BPI participants to a BPI workstep (i.e. internal Systems of Record)
- external authoritative data - data sourced from external authoritative sources (i.e. government record) and
- external non-authoritative, non-deterministic data sourced from external non-authoritative, non-deterministic sources (i.e. IoT sensor data, time, etc).

8.1 Internal Authoritative Data for BPIs

intauthbpidata

Internal authoritative data means that there exists only one authoritative version of the input data in some system of record of a BPI participant, making it the only choice for input into a BPI workstep. Authoritative in this context means that BPI participants involved in a BPI workstep have agreed that the authoritative source is the accurate and reliable truth for that particular type of data.

[O10]

Internal authoritative input data to a BPI workstep MAY come from a single source.

[O10] Testability:

Preconditions:

- A BPI test system is up and running.
- There is a given set of test BPI Subjects that are part of a test workgroup.

Non-Standards Track Work Product

- The test BPI participants in a workgroup have agreed to allow internal authoritative input data to a BPI workstep only from a single, specific data source.
- The test workgroup is associated with only one test workstep.
- The test workstep accepts only input data from the agreed-upon data source for a given data schema.
- The agreed-upon input data source is configured with a test data set.
- Another, second input data source is configured with the same test data set as the agreed-upon data source.

Test Steps:

1. Verify that the input data provided to the BPI test workstep is from the agreed-upon authoritative data source.
2. Verify that the input data complies with the established business logic of the test workstep.
3. Verify that the BPI workstep processes the input data correctly.
4. Switch the input data source, and repeat step 1.
5. Verify that the workstep creates an error, and exits with an appropriate error message.

Test Passing Criteria:

- The BPI workstep processes the first input data correctly.
- The BPI workstep creates an error, and exits with the expected error message, when the input data comes from the second input data source.

[CR36]>[O10]

If the internal authoritative input data to a BPI workstep is single-sourced, that source MUST be authoritative.

[CR36]>[O10] Testability:

Preconditions:

- A BPI test system is up and running.
- There is a given set of test BPI Subjects that are part of a test workgroup.
- The test workgroup is associated with only one test workstep.
- There exists a system of record identified as the single source for the input data.
- A test workstep has been created that requires internal authoritative input data from a single source.
- The BPI Subjects involved in the workstep have agreed that the identified system of record is the authoritative source for the input data.
- The test workstep accepts only input data from the agreed-upon data source for a given data schema.
- The agreed-upon input data source is configured with a test data set.

Test Steps:

1. Retrieve the input data for the identified BPI workstep from the single source system of record.
2. Verify that the single source system of record is the authoritative source for the input data in the test workstep.

Test Passing Criteria:

- The identified single source system of record is confirmed to be the authoritative source for the input data.

8.2 External Authoritative Data for BPIs

extauthbpidata

External authoritative data means that the input data to a BPI workstep is held in some authoritative 3rd party data store, such as government records in a centralized database. Unlike internal authoritative data where there is only one version, external authoritative data input to a BPI workstep may have multiple versions each coming from a different source such as pricing data of a commodity.

[R315]

BPI Subjects participating in a workstep MUST agree upon the source and type of the external authoritative data used as input to a BPI workstep.

[R315] Testability:

Preconditions:

- There is a given set of test BPI Subjects that are part of a test workgroup.
- The test workgroup is associated with only one test workstep.
- There exists an external system identified as the single source for the input data.
- A test workstep has been created that requires external authoritative input data from a single source.
- The test workstep accepts input data only from the agreed-upon data source for a given data schema.
- The agreed-upon input data source is configured with a test data set.
- An agreement quorum has been defined and configured in the test BPI.
- The test BPI response, if a quorum by the participants is reached, is configured.
- The test BPI response, if a quorum by the participants is not reached, is configured.

Test Steps:

1. A BPI workgroup participant initiates an agreement process among the BPI subjects in the workgroup.
2. The BPI system presents the BPI subjects with the source and type of the external authoritative data.
3. The BPI subjects review and validate the source and type of the external authoritative data.
4. If the specified quorum of BPI subjects agree, the BPI system records the agreement and uses the external authoritative data as the input source for the workstep.
5. If the quorum of BPI subject is not reached, the BPI system does not record the agreement and prompts the subjects to discuss and resolve the disagreement.
6. Once the agreement has been reached, the BPI system records the agreement and uses the external authoritative data as the input source for the workstep.
7. Start the BPI workstep that requires external authoritative data as input.
8. Complete the workstep using the agreed-upon external authoritative data.
9. Verify that the workstep was successfully completed using the agreed upon external authoritative data.

Test Passing Criteria: The test is considered passing if,

- The BPI system successfully initiates the agreement process among the BPI subjects.
- The specified quorum of BPI subjects agree upon the source and type of the external authoritative data.
 - The BPI system records the agreement and uses the agreed-upon external authoritative data as the input source for the workstep.
 - The workstep is successfully completed using the agreed-upon external authoritative data.
- The BPI system responds as configured if the specified quorum of BPI subjects on the source and type of the external authoritative data is not reached.

8.3 External Non-authoritative, Non-deterministic Data for BPIs

extnonauthbpidata

External non-authoritative, non-deterministic data means that there does not exist an authoritative, deterministic source for the data used as input to a BPI workstep. In that case, a normalizing mechanism often referred to as an oracle is needed to account for potential source discrepancies. Entities reporting external non-authoritative, non-deterministic data into a BPI worksteps may also be referred to as oracles.

Non-Standards Track Work Product

8.3.1 Data Trustworthiness

bpidatatrust

External non-authoritative, non-deterministic data for BPIs resides outside of an authoritative source and is subject to manipulation. As such, steps should be taken to remove counterparty manipulation and error risk through mechanisms such as redundancy in data reporting and error checking, either cross-party, or by an appropriately incentivized 3rd party.

[D42]

External non-authoritative, non-deterministic BPI input data into a BPI workstep SHOULD be sourced from multiple sources.

[D42] Testability:

Preconditions:

- A test BPI system is available and functional.
- At least two independent sources of external non-authoritative, non-deterministic data are available that produce the same input data.
- A test workstep has been created that requires external non-authoritative, non-deterministic input data.

Test Steps:

1. Start the test workstep that requires external non-authoritative, non-deterministic input data.
2. Source the input data from a single external source and record the source of the data.
3. Repeat step 2 for each available external source for the same input data.
4. Verify that the BPI workstep successfully processed the input data from all sources used in steps 2-3 and produced the same expected result for the input data from different sources.

Test Passing Criteria:

- The test workstep should process input data from all sources used in steps 2-3 without errors or inconsistencies.
- The expected results from the test workstep should match the actual results produced in step 4.

[R316]

External non-authoritative, non-deterministic BPI input data into a BPI workstep MUST be validated by one or more authoritative entities.

[R316] Testability:

Preconditions:

- A test BPI system is available and functional.
- A test workstep that takes external non-authoritative, non-deterministic input data.
- One or more authoritative entities responsible for validating the input data.
- The validation criteria defined and agreed upon by all BPI subjects involved in the test workstep.

Test Steps:

1. Initialize the test workstep with external non-authoritative, non-deterministic input data from one or more data sources.
2. Submit the input data to the authoritative entities responsible for validation.
3. The authoritative entities validate the input data against the predefined validation criteria.
4. If the input data passes the validation criteria, proceed:
 - a. Use the validated input data as input to the BPI workstep.
 - b. Verify that the BPI workstep produces the expected output.
5. If the input data does not pass the validation criteria, the BPI stops the workstep processing and responds to the workstep participants with an appropriate error message.

Test Passing Criteria: The test passes if,

- If the input data passes the validation criteria defined by the authoritative entities and the test workstep produces the expected output using the validated input data.
- If the input data does not pass the validation criteria defined by the authoritative entities, the BPI stops the workstep processing and responds to the workstep participants with an appropriate error message.

[R317]

BPI workstep participants MUST agree upon a validation method for external non-authoritative, non-deterministic input data to a BPI workstep.

[R317] Testability:

Preconditions:

- A test BPI system is available and functional.
- A test workstep that takes external non-authoritative, non-deterministic input data.
- There are at least two BPI workstep participants.
- Each participant is a valid BPI Subject and is authorized to participate in the test workstep.
- One or more validation methods for the external non-authoritative, non-deterministic input data to the test workstep are available to apply to the external data source to the workstep.

Test Steps:

1. One BPI workstep participant proposes one of the available validation methods for the external non-authoritative, non-deterministic input data to be used in the test workstep.
2. Each BPI workstep participant either accepts or rejects the proposal.
3. Once the validation method is agreed upon, it is recorded in the BPI.

Test Passing Criteria:

- The BPI workstep participants have agreed upon a validation method for the external non-authoritative, non-deterministic input data.
- The agreed-upon validation method has been tested and shown to produce reliable results.
- The validation method and testing results have been recorded on the test BPI.

[R318]

A BPI workstep participant MUST be able to validate that the validation criteria of external non-authoritative, non-deterministic input data to the BPI workstep has been met.

Appropriate data validation methods vary on a case-by-case basis depending upon the data types, sources and formats. As such, it is up to the BPI participants to agree upon the optimal validation method for their implementation. This includes design decisions such as what threshold of unresponsive oracles leads to a rejected input, how to aggregate the oracle responses and remove outliers, signing data to ensure provenance, implementing cryptographic mechanisms such as threshold signatures/secret sharing and TEEs such as Intel SGX or AMD SEV to obfuscate data from oracle providers.

[R318] Testability:

Preconditions:

Non-Standards Track Work Product

- A test BPI system is available and functional.
- At least one source of external non-authoritative, non-deterministic input data.
- A test workstep that takes external non-authoritative, non-deterministic input data.
- The test workstep participant has access to the external non-authoritative, non-deterministic input data.
- The validation criteria for the external non-authoritative, non-deterministic input data has been agreed upon and documented by the BPI workstep participants.
- The validation criteria for the external non-authoritative, non-deterministic input data has been implemented in the test workstep.

Test Steps:

1. Input data from an external non-authoritative, non-deterministic source.
2. Validate the external non-authoritative, non-deterministic input data against the agreed-upon validation criteria in the test workstep.
3. Record the validation result along with the validation criteria and input data.
4. Retrieve the validation result along with the validation criteria and input data
5. Validate that retrieved validation criteria and input data agree with the original validation criteria and input data and that the agreed-upon validation criteria applied to the input data match the retrieved validation result.

Test Passing Criteria:

- The validation result is recorded and indicates that the validation criteria for the external non-authoritative, non-deterministic input data have been met.
- The recorded validation criteria and input data are consistent with the agreed-upon validation method for the test workstep.
- The retrieved validation criteria and input data agree with the original validation criteria and input data and the agreed-upon validation criteria applied to the input data match the retrieved validation result.

8.3.2 External Non-authoritative, non-deterministic BPI Input Data Variance

extnonauthbpiinputdata

External non-authoritative, non-deterministic BPI input data can be subject to variations from lack of time synchronicity, fluctuations in precision or reporting error. Party A may read the temperature of a shipment as 77.1 degree Fahrenheit while party B reads it as 77.3 degree Fahrenheit. Small variance in timing can also produce mismatched BPI data inputs. These variations may lead to failure to agree on BPI workstep input data accuracy. Therefore, BPI workstep participants should agree upon their algorithmic approach to removing variance and outlier data. This agreed-upon method should have a clearly defined standard with all oracles involved adhering to the same data format and variance reduction approach.

9 Conformance

This section describes the conformance clauses and tests required to achieve an implementation that is provably conformant with the requirements in this document.

9.1 Conformance Targets

This document does not yet define a standardized set of test-fixtures with test inputs for all MUST, SHOULD, and MAY requirements with conditional MUST or SHOULD requirements.

A standardized set of test-fixtures with test inputs for all MUST, SHOULD, and MAY requirements with conditional MUST or SHOULD requirements is intended to be published with the next version of the standard.

9.2 Conformance Levels

This section specifies the conformance levels of this standard. The conformance levels offer implementers several levels of conformance. These can be used to establish competitive differentiation.

This document defines the conformance levels of a BPI as follows:

- **Level 1:** All MUST requirements are fulfilled by a specific implementation as proven by a test report that proves in an easily understandable manner the implementation's conformance with each requirement based on implementation-specific test-fixtures with implementation-specific test-fixture inputs.
- **Level 2:** All MUST and SHOULD requirements are fulfilled by a specific implementation as proven by a test report that proves in an easily understandable manner the implementation's conformance with each requirement based on implementation-specific test-fixtures with implementation-specific test-fixture inputs.
- **Level 3:** All MUST, SHOULD, and MAY requirements with conditional MUST or SHOULD requirements are fulfilled by a specific implementation as proven by a test report that proves in an easily understandable manner the implementation's conformance with each requirement based on implementation-specific test-fixtures with implementation-specific test-fixture inputs.

[D43]

A claim that a BPI conforms to this specification SHOULD describe a testing procedure carried out for each requirement to which conformance is claimed, that justifies the claim with respect to that requirement.

[D43] Testability: Since each of the non-conformance-target requirements in this documents is testable, so must be the totality of the requirements in this document. Therefore, conformance tests for all requirements can exist, and can be described as required in [D43].

[R319]

A claim that a BPI conforms to this specification at **Level 2** or higher MUST describe the testing procedure carried out for each requirement at **Level 2** or higher, that justifies the claim to that requirement.

[R319] Testability: Since each of the non-conformance-target requirements in this documents is testable, so must be the totality of the requirements in this document. Therefore, conformance tests for all requirements can exist, be described, be built and implemented and results can be recorded as required in [R319].

Note that BPI Integration requirements in section 5.5.4 Bi- and Multi-directional and Mono-directional BPI Interoperability Services and section 5.6 Standardized Set of BPI Interoperability APIs are not mandatory for meeting conformance until there are at least two implementations conformant to Level 1 of this standards' requirements.

Appendix A - References

This appendix contains the normative and non-normative references that are used in this document.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

A.1 Normative References

The following documents are referenced in such a way that some or all of their content constitute requirements of this document.

[RFC2119]

S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

[NIST-SP-800-32]

NIST SP 800-32, 2001, <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-32.pdf>.

[NIST-SP-1800-21B]

Non-Standards Track Work Product

NIST SP 1800-21B, 2020, <https://doi.org/10.6028/NIST.SP.1800-21>.

[NIST-SP-800-192]

NIST SP 800-192, 2017, <https://doi.org/10.6028/NIST.SP.800-192>.

[W3C-DID]

Decentralized Identifiers (DIDs) v1.0, M. Sporny, D. Longley, M. Sabadello, D. Reed, O. Steele, C. Allen, W3C Proposed Recommendation, August 2021 <https://www.w3.org/TR/2021/PR-did-core-20210803/>. Latest version available at <https://www.w3.org/TR/did-core/>.

[W3C-VC]

Verifiable Credentials Data Model 1.0, Manu Sporny, Dave Longley, David Chadwick, W3C Recommendation, November 2019 <https://www.w3.org/TR/2019/REC-vc-data-model-20191119/>. Latest version available at <https://www.w3.org/TR/vc-data-model/>.

[ISO-IEC-27033]

ISO/IEC 27033: Information technology — Security techniques — Network security - Parts 1 through 6 published by ISO.

[RFC3339]

Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002 <https://www.rfc-editor.org/info/rfc3339>.

[RFC5246]

Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008 <https://www.rfc-editor.org/info/rfc5246>.

[RFC8446]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018 <https://www.rfc-editor.org/info/rfc8446>.

[RFC2818]

Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC7516]

Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015 <https://www.rfc-editor.org/info/rfc7516>.

[RFC7515]

Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015 <https://www.rfc-editor.org/info/rfc7515>.

[RFC7519]

Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015 <https://www.rfc-editor.org/info/rfc7519>.

[RFC8446]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018 <https://www.rfc-editor.org/info/rfc8446>.

[RFC7159]

Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014 <https://www.rfc-editor.org/info/rfc7159>.

[JSONLD]

JSON-LD 1.1, M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, Pierre-Antoine Champin, N. Lindström, W3C Recommendation, July 2020 <https://www.w3.org/TR/2020/REC-json-ld11-20200716/>. Latest version available at <https://www.w3.org/TR/json-ld11/>.

A.2 Non-Normative References

[CVMP]

NIST CVMP, <https://csrc.nist.gov/projects/cryptographic-module-validation-program>.

[FIPS]

FIPS, <https://www.nist.gov/itl/current-fips>.

[ISO27001]

ISO/IEC 27001:2013, <https://www.iso.org/standard/54534.html>.

[whois]

ICANN, Domain Name Registration, <https://whois.icann.org/en/domain-name-registration-process>

[X509]

International Telecommunications Union, October 2019, <https://www.itu.int/rec/T-REC-X.509-201910-I/en>.

[CA]

NIST SP 800-56B Rev. 2, March 2019, <https://doi.org/10.6028/NIST.SP.800-56Br2>.

[How-to-Explain-Zero-Knowledge-Protocols-to-Your-Children]

Quisquater, Jean-Jacques; Guillou, Louis C.; Berson, Thomas A. (1990). "How to Explain Zero-Knowledge Protocols to Your Children". Advances in Cryptology – CRYPTO '89: Proceedings. Lecture Notes in Computer Science. 435. pp. 628–631. doi:10.1007/0-387-34805-0_60. ISBN 978-0-387-97317-3.

[The-Byzantine-Generals-Problem]

"The Byzantine Generals Problem", Leslie Lamport, Robert E. Shostak, Marshall Pease, ACM Transactions on Programming Languages and Systems, 1982.

[OAuth-2.0]

Aaron Parecki, (2020), "OAuth 2.0 Simplified", ISBN-13: 978-1387751518.

[SAML]

Non-Standards Track Work Product

J. Hughes et al. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, March 2005. Document identifier: saml-profiles-2.0-os.

[w3c-holder-definition]

W3C Verifiable Credential Data Model: Holder Definition, <https://www.w3.org/TR/2022/REC-vc-data-model-20220303/#dfn-holders> (2022).

[OIDC]

OpenID Connect Federation 1.0, <https://openid.net/developers/specs/> (2019).

[api-portal]

https://github.com/api-evangelist/portal-minimum/blob/gh-pages/_config.yml (2018).

[ActiveDirectory]

"Directory System Agent". MSDN Library. Microsoft. (2018).

[SSO]

Recommendations of the National Institute of Standards and Technology (NIST Guide to Secure Web Services). NIST SP 800-95, <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-95.pdf>.

[HSM]

Ramakrishnan, Vignesh; Venugopal, Prasanth; Mukherjee, Tuhin (2015). Proceedings of the International Conference on Information Engineering, Management and Security 2015: ICIEMS 2015. Association of Scientists, Developers and Faculties (ASDF). p. 9. ISBN 9788192974279.

[libp2p]

libp2p, <https://docs.ipfs.io/concepts/libp2p/>

[CRDT]

Shapiro, Marc; Pregoça, Nuno; Baquero, Carlos; Zawirski, Marek (2011), "Conflict-Free Replicated Data Types", Lecture Notes in Computer Science, 6976, Grenoble, France: Springer Berlin Heidelberg, pp. 386–400, doi:10.1007/978-3-642-24550-3_29.

[ITC]

Almeida P.S., Baquero C., Fonte V. (2008) Interval Tree Clocks. In: Baker T.P., Bui A., Tixeuil S. (eds) Principles of Distributed Systems. OPODIS 2008. Lecture Notes in Computer Science, vol 5401. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-92221-6_18

[DHT]

Liz, Crowcroft; et al. (2005). "A survey and comparison of peer-to-peer overlay network schemes" (PDF). IEEE Communications Surveys & Tutorials. 7 (2): 72–93. doi:10.1109/COMST.2005.1610546

[FPDS]

Driscoll JR, Sarnak N, Sleator DD, Tarjan RE (1986). "Making data structures persistent". Proceedings of the eighteenth annual ACM symposium on Theory of computing - STOC '86. Proceeding STOC '86. Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing. pp. 109–121. CiteSeerX 10.1.1.133.4630. doi:10.1145/12130.12142. ISBN 978-0-89791-193-1.

[PPDS]

Conchon, Sylvain; Filliâtre, Jean-Christophe (2008), "Semi-persistent Data Structures", Programming Languages and Systems, Lecture Notes in Computer Science, 4960, Springer Berlin Heidelberg, pp. 322–336, doi:10.1007/978-3-540-78739-6_25, ISBN 9783540787389.

[DANE]

Barnes, Richard (October 6, 2011). "DANE: Taking TLS Authentication to the Next Level Using DNSSEC", IETF Journal.

[Shamir]

Shamir, Adi (1979), "How to share a secret", Communications of the ACM, 22 (11): 612–613 doi:10.1145/359168.359176, S2CID 16321225.

[NATS]

NATS, version 2.2 (2021), <https://docs.nats.io/nats-concepts/intro>

[amqp-core-complete-v1.0]

OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. 29 October 2012. OASIS Standard. <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf>

[DIDCOMM]

Decentralized Identity Foundation, DIDComm Messaging Editor's Draft, <https://identity.foundation/didcomm-messaging/spec/>

[SIOP]

Decentralized Identity Foundation, Self-Issued OpenID Connect Provider DID Profile v0.1, <https://identity.foundation/did-siop/>

A.3 Example References

[Functional-Terms-Implemented-as-Zero-Knowledge-Circuit]

```
import "utils/pack/u32/nonStrictUnpack256" as unpack256
import "utils/pack/u32/unpack128" as unpack128
import "hashes/sha256/512bitPacked" as packed512Sha
import "ecc/edwardsCompress" as edwardsCompress
import "hashes/sha256/756bit" as shaOf756Bits
import "utils/casts/bool_256_to_u32_8" as bool_256_to_u32_8
import "ecc/edwardsScalarMult" as multiply
import "utils/pack/bool/nonStrictUnpack256" as unpack256Bool
import "ecc/babyjubjubParams" as context
from "ecc/babyjubjubParams" import BabyJubJubParams
```

```
struct Commitment {
  field[2] value
```

Non-Standards Track Work Product

```
field[2] salt
}

struct Metadata {
  field[2] senderPublicKey
  field agreementName
}

// for example hash could be the hash of the N30 payment term
struct PaymentAgreement {
  u32[8] hash
  field[2] senderPublicKey
}

def convert(field[2] input) -> u32[8]:
  u32[4] lsbBits = unpack128(input[0])
  u32[4] msbBits = unpack128(input[1])
  return [...msbBits, ...lsbBits]

def main(field publicInputHash, field[2] pk, private Commitment inputCommitment, private Metadata inputMetadata, private field sk) -> PaymentAgreement:
  u32[8] publicInputHashBits = unpack256(publicInputHash)

  // hash of the below private inputs must be equal to public input
  // Convert inputs to u32
  u32[8] saltBits = convert(inputCommitment.salt)
  u32[8] valueBits = convert(inputCommitment.value)
  bool[256] compressedSenderPubKey = edwardsCompress(inputMetadata.senderPublicKey)
  u32[8] pubKeyBits = bool_256_to_u32_8(compressedSenderPubKey)
  u32[8] nameBits = unpack256(inputMetadata.agreementName)

  // compute hash
  u32[8] paymentAgreementHash = shaOf756Bits(saltBits, pubKeyBits, nameBits)

  // Check: Compare final hash to public input hash; and Compare input
  bool out = publicInputHashBits == paymentAgreementHash && valueBits == paymentAgreementHash

  // Check: Prove ownership of agreement
  BabyJubJubParams ctx = context()
  field[2] G = [ctx.Gu, ctx.Gv]
  bool[256] skBits = unpack256Bool(sk)
  field[2] ptExp = multiply(skBits, G, ctx)
  bool owned = ptExp[0] == pk[0] && ptExp[1] == pk[1]

  // Prepare output object
  PaymentAgreement outputAgreement = PaymentAgreement { hash: paymentAgreementHash, senderPublicKey: inputMetadata.senderPublicKey }

  return outputAgreement
```

[Storing-an-Agreement-as-State-Object-in-Merkle-Tree]

Example of Storing an Agreement as a state object in a Merkle Tree using the Payment Term agreement as an example:

Agreement and Public Input Data Leafs
Leaf 1: H({AgreementType: PaymentTerm})
Leaf 2: H({AgreementID: A1D324BFCE})
Leaf 3: H({AgreementDate: 1627601020})
Leaf 3: H({PaymentTerm: 30})
Leaf 4: H({SigningKey1: pk_buyer})
Leaf 5: H({SigningKey1: pk_seller})

Root is calculated normally.

Agreement Proof Leafs:

Leaf1: H({DocumentType: Invoice, Proof: zk-proof_Invoice1, PublicInput: Struct of the Leafs})
Leaf 2: H({DocumentType: Invoice, Proof: zk-proof_Invoice1, PublicInput: Struct of the Leafs})

Root is calculated normally.

Optionally, the two trees can be joined into a 3rd full agreement state Merkle Tree. A secure, offchain Merkle Tree library can be found [here](#).

Example of Merkle Proof Verification Circuit to validate a State Object:

// ABOUT

// Function for proving membership of a leaf in a Merkle Tree of height h = 4.

```
//
//   level h:   root
//             /  \
//            ...
//           ...
//          /  \  /  \
//         level 1:
//        /  \  /  \  /  \
//       leaves at level 0:
```

// IMPORTS

```
import "hashes/sha256/512bit.zok" as sha256of512
import "../hashes/sha256/padding/shaPad432To512.zok" as shaPad432To512

import "utils/pack/bool/unpack128.zok" as unpack128
import "../packing/unpack1x216To216x1.zok" as unpack1x216To216x1
```

Non-Standards Track Work Product

```
import "../packing/unpack2x128To256x1.zok" as unpack2x128To256x1

import "../concatenate/orderedConcatenate216x216.zok" as orderedConcatenate216x216

// MAIN

// @param {field[4]} siblingPath - the values of the sibling nodes of the path from the leaf to the root. Assume each field is 216-bits.
// @param {field[2]} leafValue - the value of the leaf. We aim to prove this leaf is in the tree.
// @param {field} leafIndex - the index of the leaf within the leaves (indexing starts from zero). Note: the binary decomposition of a leaf's index gives us the 'left-rightness' of that leaf's path up the Merkle Tree.
// @returns {field} root - the root of the merkle tree

def main(private field[4] siblingPath, private field[2] leafValue, private field leafIndex) -> field[256]:

  // Hash up the Merkle Tree to the root:

  field[128] leafIndexBits = unpack128(leafIndex)
  field[256] nodeValueBits = unpack2x128To256x1(leafValue)

  for field i in 0..4 do
    field j = 3 - i // iterator for the siblingPath
    field k = 127 - i // iterator for the leafIndexBits

    field[216] siblingNodeValueBits = unpack1x216To216x1(siblingPath[j])

    field[432] preimage432 = orderedConcatenate216x216(leafIndexBits[k], nodeValueBits[40..256], siblingNodeValueBits)

    field[512] preimage512 = shaPad432To512(preimage432)

    nodeValueBits = sha256of512(preimage512[0..256], preimage512[256..512])
  endfor

  field[256] root = nodeValueBits

  return root
```

[Commercial-Agreement-as-Verifiable-Credential]

An example implementation of a signed Commercial Agreement using a verifiable credential. The VC below, showing an order between a buyer and seller, serves as input into a state object representing the Order. This example shows the binding contract between the buyer and the seller, the seller requesting the verification of the product order prior to its acceptance, and the buyer accepting the order if the functional terms are valid.

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/traceability/v1"
  ],
  "id": "http://example.org/credentials/",
  "type": [
    "VerifiableCredential"
  ],
  "issuanceDate": "2021-02-04T20:29:37+00:00",
  "issuer": { "did:key:z6MktHQo3fRRohk44dsbE76CuiTpBmyMWq2VVjvV6aBSeE3U",
  "credentialSubject": {
    "@context": [
      "https://w3id.org/traceability/v1"
    ],
    "type": "EcommerceOrderRegistrationCredential",
    "buyer": { "did:example:123",
    "orderID": "Order#975",
    "productInOrder": [
      "https://vc.example.com/?queryID=6206f1f744a781480c521902a1a1dbf5f1d01e7ea21daf483e7668817e58598a",
      "https://vc.example.com/?queryID=6206f1f744a781480c521902a1a1dbf5f1d01e7ea21daf483e7668817e58598a"
    ],
    "certificateName": "ACME Ecommerce Order Registration Certificate"
  },
  "evidence": [
    {
      "type": [
        "DocumentVerificationEvidence"
      ],
      "id": "https://example.acme.com/evidence/?queryID=0xFd5FEB812fFa20bEBDcBCD63dC11e96A7A1D59c14fAbEAF9c55D006Ac9DEac3B",
      "verifier": {
        "did:web:www.acme.com"
      },
      "evidenceDocument": "ACME-Evidence-Document-0x2b440EbE-2-4-2021",
      "subjectPresence": "InPerson",
      "documentPresence": "Digital"
    }
  ],
  "credentialStatus": {
    "type": [
      "RevocationList2020Status"
    ],
    "id": "https://example.acme.com/credential/status/?queryID=0xFd5FEB812fFa20bEBDcBCD63dC11e96A7A1D59c14fAbEAF9c55D006Ac9DEac3B#23323",
    "revocationListIndex": "23323",
    "revocationListCredential": "https://example.com/credentials/status/?queryID=0xFd5FEB812fFa20bEBDcBCD63dC11e96A7A1D59c14fAbEAF9c55D006Ac9DEac3B"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "created": "2019-12-11T03:50:55Z",
    "jws": "eyJhbGciOiJIJzERTQSIslml2NCi6ZmFsc2UsImNyaXQiOlsiYjY0Il19..4vwwJBA0mXPn2R7g_zCNCJ8qgJAAsLhrsU9QJzrJPPQrRh7JiQr-NespMPlg36A9TyIn6uP67WMqPhFFBYMDNBQ",
    "proofPurpose": "assertionMethod",
```

Non-Standards Track Work Product

```
"verificationMethod": "did:key:z6MktHQo3fRRohk44dsbE76CuiTpBmyMWq2VVjvV6aBSeE3U#z6MktHQo3fRRohk44dsbE76CuiTpBmyMWq2VVjvV6aBSeE3U"
}, {
  "type": "BbsBlsSignature2020",
  "created": "2021-05-21T15:31:30Z",
  "proofPurpose": "assertionMethod",
  "proofValue":
"ky+cRs+7xIw4FxccF16E5g9HjvbdochfiBkITba37+xomLAAHcv8nza1PK0Y/ux7XeULTDrrhbw2mFGk3AHqRQtH4yRIZBP1fOZDI8E8KRC2xRILq6v4xrzy/CFLV7QdRpaqMJ4o8A3WmXGlxfLA==",
  "verificationMethod": "did:example:123#key-1"
}]
}
```

[Ethereum-Client-Transaction-Crafting-Function]

```
import { ethers } from 'ethers';
import { ITxManager } from '.';
import { logger } from './logger';
import { jsonrpc, shieldContract } from './blockchain';
```

```
export class EthClient implements ITxManager {
  constructor(public signer: any, public signerType: string) {
    this.signerType = signerType;
    this.signer = signer;
  }
}
```

```
async constructTx(toAddress: string, fromAddress: string, txData: string) {
  logger.debug('Received request for EthClient.signTx');
  const { result: nonce } = await jsonrpc('eth_getTransactionCount', [
    process.env.WALLET_PUBLIC_KEY,
    'latest'
  ]);
  logger.debug('nonce: ${nonce}');
  const { result: gasPrice } = await jsonrpc('eth_gasPrice', []);
  logger.debug('gasPrice found: ${gasPrice}');
  const gasPriceSet = Math.ceil(Number(gasPrice) * 1.2);
  logger.debug('gasPrice set: ${gasPriceSet}');
```

```
const unsignedTx = {
  to: toAddress || '',
  from: fromAddress,
  data: txData,
  nonce,
  chainId: parseInt(process.env.CHAIN_ID, 10),
  gasLimit: 0,
  gasPrice: '0x' + gasPriceSet.toString(16)
};
```

```
// key-manager returns 400 if "from" field is provided in tx
if (this.signerType === 'key-manager') {
  delete unsignedTx.from;
}
```

```
const res = await jsonrpc('eth_estimateGas', [unsignedTx]);
const gasEstimate = res.result;
logger.debug('gasEstimate: ${gasEstimate}');
if (!gasEstimate) {
  return {
    error: {
      code: -32000,
      message: 'eth_estimateGas returned null value'
    }
  };
}
unsignedTx.gasLimit = Math.ceil(Number(gasEstimate) * 1.1);
logger.debug('gasLimit set: ${unsignedTx.gasLimit}');
```

```
logger.debug('Unsigned tx: ' + JSON.stringify(unsignedTx, null, 4));
const signedTx = await this.signer.signTransaction(unsignedTx, fromAddress);
logger.debug('Signed tx: ${signedTx}');
return { result: signedTx };
}
```

```
async sendTransaction(toAddress: string, fromAddress: string, txData: string) {
  logger.debug('Received request for EthClient.sendTransaction');
  let error = null;
  let txHash: string;
  try {
    const { error: constructError, result: signedTx } = await this.constructTx(
      toAddress,
      fromAddress,
      txData
    );
    if (constructError) {
      return { error: constructError };
    }
    const res = await jsonrpc('eth_sendRawTransaction', [signedTx]);
    txHash = res.result;
  }
```

Non-Standards Track Work Product

```
    } catch (err) {
      logger.error('EthClient.sendTransaction:', err);
      if (err.error) {
        error = { data: err.error.message };
      } else {
        error = { data: err };
      }
    }
    return { error, txHash };
  }
}
```

Appendix B - Security Considerations

There are no additional security requirements.

It should be noted that any BPI should have completed a security audit by a reputable security auditor and resolved all security issues before going to production.

B.1 Data Privacy

The standard does not set any requirements for compliance to jurisdiction legislation/regulations, the responsibility of the implementer to comply with applicable data privacy laws.

B.2 Production Readiness

The standard does not set any requirements for the use of specific applications/tools/libraries etc. The implementer should perform due diligence when selecting specific applications/tools/libraries.

B.3 Internationalization and Localization Reference

The standard encourages implementers to follow the [W3C "Strings on the Web: Language and Direction Metadata" best practices guide](#) for identifying language and base direction for strings used on the Web wherever appropriate.

Appendix C - Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged.

Participants:

Andreas Freund
Anais Ofranc, Consianimis
Gage Mondok, Chainlink
Kyle Thomas, Provide
Daven Jones, Provide
Mehran Shakeri, SAP
Alessandro Gasch, SAP
John Wolpert, ConsenSys
Sam Stokes, ConsenSys
Nick Kritikos, ConsenSys
Keith Salzman, ConsenSys
Yoav Bittan, ConsenSys
Kailen Patel
Chaals Neville, EEA

Appendix D - Revision History

Revisions made since the initial stage of this numbered Version of this document have been tracked on [Github](#).

Appendix E - Notices

Copyright © OASIS Open 2024. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This specification is published under the [CC0 1.0 Universal \(CC0 1.0\)](#) license. Portions of this specification are also provided under the [Apache License 2.0](#).

All contributions made to this project have been made under the [OASIS Contributor License Agreement \(CLA\)](#).

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the [Open Projects IPR Statements](#) page.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restrictions of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Open Project (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OASIS AND ITS MEMBERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE

Non-Standards Track Work Product

OF THIS DOCUMENT OR ANY PART THEREOF.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Project Specifications, OASIS Standards, or Approved Errata).

[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Open Project that produced this deliverable.]

[OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Open Project that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website but disclaims any obligation to do so.]

[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this OASIS Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Open Project can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]

The name "OASIS" is a trademark of [OASIS](https://www.oasis-open.org), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation, and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <https://www.oasis-open.org/policies-guidelines/trademark> for the above guidance.