

# Enterprise Ethereum Alliance Client Specification v4

8 October 2019



**Latest editor's draft:**

<https://entethalliance.github.io/client-spec/spec.html>

## **Editors:**

[Robert Coote](#) (PegaSys)

[Chaals Nevile](#) (Enterprise Ethereum Alliance)

[Grant Noble](#) (PegaSys)

[George Polzer](#) (Everymans.ai)

## **Former editors:**

Daniel Burnett ([PegaSys](#))

David Hyland-Wood ([PegaSys](#))

## **Contributors to this version:**

Duarte Aragao (Clearmatics), Janie Baños (Dekra), Sanjay Bakshi (Intel), Imran Bashir (Quorum), Meredith Baxter (PegaSys), Tim Beiko (PegaSys), Tomas Bertani (ProvableThings), Coogan Brennan (ConsenSys), Mark Bruening (BakerHostetler), Benjamin Burns (Whiteblock), Jean-Charles Cabalguen (iExec), David Clark (Truffle), Zak Cole (Whiteblock), Rob Dawson (PegaSys), Anthony Denyer (Web3Labs), Paul DiMarzio (EEA), Dan Doney (Securrency), Samer Falah (Quorum), Sara Feenan (Clearmatics), Danno Ferrin (PegaSys), Andreas Freund (ConsenSys), Ken Fromm (EEA), Eduardo Garcia (Accenture), Puneetha Karamsetty (Web3Labs), Libby Kent (Quorum), Ivaylo Kirilov (Web3Labs), Maya Konaka (Blockapps), Kieren James-Lubin (Blockapps), Tom Lindeman (ConsenSys), Tyrone Lobban (Quorum), Chris McKay (PegaSys), Arash Mahboubi (PegaSys), Boris Mann (SPADE), Madeline Murray (PegaSys), George Ornbo (Clearmatics), Eric Rafaloff (TrailOfBits), Brianna Rich (EEA), Peter de Rooij (Accenture), Lior Saar (BlockApps), Joseph Schweitzer (Ethereum Foundation), Felix Shnir (JP Morgan Chase), Przemek Siemion (Banco Santander), Conor Svensson (Web3Labs), Clark Thompson (ConsenSys), Antoine Toulme (Whiteblock), Tom Willis (Intel), Victor Wong (BlockApps), Yevgeniy 'Eugene' Yarmosh (Intel), Lei Zhang (iExec), Jim Zhang (ConsenSys), Weijia Zhang (Wanchain)

## Abstract

This document, the Enterprise Ethereum Alliance Client Specification, defines the implementation requirements for Enterprise Ethereum clients, including the interfaces to external-facing components of Enterprise Ethereum and how they are intended to be used.

## Copyright, Licensing, and Limitation of Warranty

The copyright in this document is owned by Enterprise Ethereum Alliance Inc. (“EEA” or “Enterprise Ethereum Alliance”). No modifications, edits or changes to the information in this document are permitted. Subject to the terms and conditions described herein, this document may be duplicated for internal use, provided that all copies contain all proprietary notices and disclaimers included herein. Except as otherwise provided herein, no license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

Use of this document and any related intellectual property incorporated herein, is also governed by the Bylaws, Intellectual Property Rights Policy and other governing documents and policies of EEA and is subject to the disclaimers and limitations described below.

No use or display of any of the following names or marks “Enterprise Ethereum Alliance”, the acronym “EEA”, the EEA logo or any combination thereof) to claim compliance with or conformance to this document (or similar statements) is permitted absent EEA membership and express written permission from the EEA. The EEA is in process of developing a conformance testing and certification program only for the EEA members in good standing, which it targets to launch towards the end of 2020.

THE CONTENTS OF THIS DOCUMENT ARE PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, SATISFACTORY QUALITY, OR REASONABLE SKILL OR CARE, OR ANY WARRANTY ARISING OUT OF ANY COURSE OF DEALING, USAGE, TRADE PRACTICE, PROPOSAL, SPECIFICATION OR SAMPLE. EEA DOES NOT WARRANT THAT THIS DOCUMENT IS COMPLETE OR WITHOUT ERROR AND DISCLAIMS ANY WARRANTIES TO THE CONTRARY.

Each user of this document hereby acknowledges that software or products implementing the technology specified in this document (“EEA-Compliant Products”) may be subject to various regulatory controls under the laws and regulations of various governments worldwide. Such laws and regulatory controls may govern, among other things, the combination, operation, use, implementation and distribution of EEA-Compliant Products. Examples of such laws and regulatory controls include, but are not limited to, airline regulatory controls, telecommunications regulations, finance industry and security regulations, technology transfer controls, health and safety and other types of regulations. Each user of this document is solely responsible for the compliance by their EEA-Compliant Products with any such laws and regulations and for obtaining any and all required authorizations, permits, or licenses for their EEA-Compliant Products related to such regulations within the applicable jurisdictions. Each user of this document acknowledges that nothing in this document or the relevant specification provides any information or assistance in connection with securing such compliance, authorizations or licenses. NOTHING IN THIS DOCUMENT CREATES ANY WARRANTIES WHATSOEVER REGARDING THE APPLICABILITY OR NON-APPLICABILITY OF ANY SUCH LAWS OR REGULATIONS OR THE SUITABILITY OR NON-SUITABILITY OF ANY SUCH PRODUCT OR SERVICE FOR USE IN ANY JURISDICTION.

EEA has not investigated or made an independent determination regarding title or non-infringement of any technologies that may be incorporated, described or referenced in this document. Use of this document or implementation of any technologies described or referenced herein may therefore infringe undisclosed third-party patent rights or other intellectual property rights. The user is solely responsible for making all assessments relating to title and non-infringement of any technology, standard, or specification referenced in this document and for obtaining appropriate authorization to use such technologies, standards, and specifications, including through the payment of any required license fees.

NOTHING IN THIS DOCUMENT CREATES ANY WARRANTIES OF TITLE OR NONINFRINGEMENT WITH RESPECT TO ANY TECHNOLOGIES, STANDARDS OR SPECIFICATIONS REFERENCED OR INCORPORATED INTO THIS DOCUMENT. IN NO EVENT SHALL EEA OR ANY OF ITS MEMBERS BE LIABLE TO THE USER OR TO A THIRD PARTY FOR ANY CLAIM ARISING FROM OR RELATING TO THE USE OF THIS DOCUMENT, INCLUDING, WITHOUT LIMITATION, A CLAIM THAT SUCH USE INFRINGES A THIRD PARTY'S INTELLECTUAL PROPERTY RIGHTS OR THAT IT FAILS TO COMPLY WITH APPLICABLE LAWS OR REGULATIONS. BY USE OF THIS DOCUMENT, THE USER WAIVES ANY SUCH CLAIM AGAINST EEA AND ITS MEMBERS RELATING TO THE USE OF THIS DOCUMENT. EEA reserves the right to adopt any changes or alterations to this document as it deems necessary or appropriate without any notice. User is solely responsible for determining whether this document has been superseded by a later version or a different document.

## Status of This Document

*This section describes the status of this document at the time of its publication. Newer documents might supersede this document.*

This is a Board Review draft of the Enterprise Ethereum Alliance Client Specification version 4. [Changes made](#) since version 3 of the Specification, released on 13 May 2019, have been reviewed by the Enterprise Ethereum Alliance (EEA) Technical Specification Working Group (TSWG) but not the EEA Board.

The TSWG propose this revision of the Specification to the EEA Board for approval and publication on 10 October 2019, obsoleting version 3. The TSWG **expects** to produce a further revision of this specification for release in the second quarter of 2020.

Although predicting the future is known to be difficult, as well as ongoing quality enhancement, future work on this Specification is expected to include the following aspects:

- Private transaction implementation.
- Agree on a BFT consensus algorithm.
- Offchain and trusted computing APIs.
- Cross-chain interoperability.
- Tracking developments for Eth 1.x and Eth 2.0.
- Requirements for Enterprise Ethereum blockchains.

## Table of Contents

<b>1.</b>	<b>Introduction</b>
1.1	Why Produce a Client Specification?
<b>2.</b>	<b>Conformance</b>
2.1	Experimental Requirements
2.2	Requirement Categorization
<b>3.</b>	<b>Security Considerations</b>

- 3.1      Callback URL Sanitization
- 3.2      Attacks on Enterprise Ethereum
- 3.3      Positive Security Design Patterns
- 3.4      Handling of Sensitive Data
- 3.5      Security of Client Implementations
  
- 4.      Enterprise Ethereum Architecture**
  
- 5.      Application Layer**
- 5.1      ÐApps Sublayer
- 5.2      Infrastructure Contracts and Standards Sublayer
- 5.3      Smart Contract Tools Sublayer
  
- 6.      Tooling Layer**
- 6.1      Credential Management Sublayer
- 6.2      Integration and Deployment Tools Sublayer
  - 6.2.1      Enterprise Management Systems
- 6.3      Client Interfaces and APIs Sublayer
  - 6.3.1      Compatibility with the Core Ethereum JSON-RPC API
  - 6.3.2      Extensions to the JSON-RPC API
    - 6.3.2.1      eea\_sendTransactionAsync
    - 6.3.2.2      eea\_sendTransaction
    - 6.3.2.3      eea\_sendRawTransaction
    - 6.3.2.4      eea\_sendRawTransactionAsync
  - 6.3.3      Permissioning Smart Contract
    - 6.3.3.1      Permissioning enforcement
    - 6.3.3.2      Permissioning management
    - 6.3.3.3      Node Permissioning
      - 6.3.3.3.1      Node Permissioning Functions
      - 6.3.3.3.2      Node Permissions
      - 6.3.3.3.3      Client Implementation
      - 6.3.3.3.4      Chain Initialization
    - 6.3.3.4      Account Permissioning
      - 6.3.3.4.1      Account Permissioning Function
      - 6.3.3.4.2      Client Implementation
      - 6.3.3.4.3      Contract Implementation
      - 6.3.3.4.4      Chain Initialization
  - 6.3.4      Inter-chain
  - 6.3.5      Oracles
  
- 7.      Enterprise 3 P's Layer**
- 7.1      Privacy Sublayer

7.1.1	On-chain
7.1.2	Off-chain (Trusted Computing)
7.1.3	Private Transactions
7.1.4	Privacy Groups
7.2	Performance Sublayer
7.2.1	On-chain (Layer 1 and Layer 2) Scaling
7.2.2	Off-chain (Layer 2 Compute)
7.3	Permissioning Sublayer
7.3.1	Nodes
7.3.2	Ethereum Accounts
7.3.3	Additional Permissioning Requirements
<b>8.</b>	<b>Core Blockchain Layer</b>
8.1	Storage and Ledger Sublayer
8.2	Execution Sublayer
8.2.1	Finality
8.3	Consensus Sublayer
<b>9.</b>	<b>Network Layer</b>
9.1	Network Protocol Sublayer
<b>10.</b>	<b>Anti-spam</b>
<b>11.</b>	<b>Cross-client Compatibility</b>
<b>12.</b>	<b>Cross-chain Interoperability</b>
<b>13.</b>	<b>Synchronization and Disaster Recovery</b>
<b>A.</b>	<b>Additional Information</b>
A.1	Terms defined in this specification
A.2	Events, functions, methods, parameters defined in this specification
A.3	Summary of Requirements
A.4	Acknowledgments
A.5	Changes
<b>B.</b>	<b>References</b>
B.1	Normative references
B.2	Informative references

## 1. Introduction

*This section is non-normative.*

This document, the Enterprise Ethereum Alliance Client Specification, defines the implementation requirements for [Enterprise Ethereum clients](#), including the interfaces to external-facing components of [Enterprise Ethereum](#) and how they are intended to be used. A [partial list of use cases](#) that this specification attempts to address is available as a work in progress [[USECASES](#)].

For the purpose of this Specification:

- **Public Ethereum** (Ethereum) is the public blockchain-based distributed computing platform featuring [smart contract](#) (programming) functionality defined by the [[Ethereum-Yellow-Paper](#)], [[EIPs](#)], and associated specifications.
- **Ethereum MainNet** (MainNet) is the [public Ethereum](#) blockchain whose [chainid](#) and [network ID](#) are both **1**.
- **Enterprise Ethereum** is the set of enterprise-focused extensions to [public Ethereum](#) defined in this Specification. These extensions provide the ability to perform [private transactions](#), and enforce [permissioning](#), for [Ethereum](#) blockchains that use them. Such blockchains are known as **Enterprise Ethereum blockchains**.
- An **Enterprise Ethereum client** (a client) is the software that implements [Enterprise Ethereum](#), and is used to run [nodes](#) on an [Enterprise Ethereum blockchain](#).
- A **node** is an instance of an [Enterprise Ethereum client](#) running on an [Enterprise Ethereum blockchain](#).

#### NOTE

Multiple [clients](#) might run on an individual device, or a [client](#) might run on a cloud service.

## 1.1 Why Produce a Client Specification?

With a growing number of vendors developing [Enterprise Ethereum clients](#), meeting the requirements outlined in this Client Specification ensures different [clients](#) can communicate with each other and **interoperate** reliably on a given [Enterprise Ethereum blockchain](#).

For [DApp](#) developers, for example, a Client Specification ensures [clients](#) provide a set of identical interfaces so that [DApps](#) will work on all conforming [clients](#). This enables an ecosystem where users can change the software they use to interact with a running blockchain, instead of being forced to rely on a single vendor to provide support.

From the beginning, this approach has underpinned the development of [Ethereum](#), and it meets a key need for blockchain use in many enterprise settings.

Client diversity also provides a natural mechanism to help verify that the protocol specification is unambiguous because interoperability errors revealed in development highlight parts of the protocol that different engineering teams interpret in different ways.

Finally, standards-based interoperability allows users to leverage the widespread knowledge of Ethereum in the blockchain development community to minimize the learning curve for working with Enterprise Ethereum, and thus reduces risk when deploying an Enterprise Ethereum blockchain.

## 2. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *MUST NOT*, *SHOULD*, and *SHOULD NOT* in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 2.1 Experimental Requirements

This Specification includes requirements and Application Programming Interfaces (APIs) that are described as *experimental*. Experimental means that a requirement or API is in early stages of development and might change as feedback is incorporated. Implementors are encouraged to implement these experimental requirements, with the knowledge that requirements in future versions of the Specification are not guaranteed to be compatible with the current version. Please send your comments and feedback on the experimental portions of this Specification to the EEA Technical Steering Committee at <https://entethalliance.org/contact/>.

### 2.2 Requirement Categorization

All requirements in this Specification are categorized as either:

- ***Protocol requirements***, denoted by [P] prefixed to the requirement ID.

Protocol requirements are requirements where the desired properties and correctness of the system can be jeopardized unless all clients implement the requirement correctly.

- ***Client requirements***, denoted by [C] prefixed to the requirement ID.

Client requirements do not impact global system behavior, but if not implemented correctly in a client, that client might not function correctly, or to a desirable level, in an Enterprise



[Ethereum](#) blockchain.

#### EXAMPLE 1: Requirement Categorization

**[P] SMRT-030:** [Enterprise Ethereum clients](#) *MUST* support [smart contracts](#) of at least 24,576 bytes in size.

Requirement **SMRT-030** is a protocol requirement. Running a [client](#) that does not implement this requirement on an [Enterprise Ethereum blockchain](#) risks causing an error in the functioning of the blockchain.

**[C] JRPC-050:** [Enterprise Ethereum clients](#) *MUST* implement the [\[JSON-RPC-PUB-SUB\]](#) API.

Requirement **JRPC-050** is a [client](#) requirement, which if not implemented correctly, does not disrupt the correct functioning of an [Enterprise Ethereum blockchain](#).

## 3. Security Considerations

*This section is non-normative.*

Security of information systems is a major field of work. [Enterprise Ethereum](#) software development shares with all software development the need to consider security issues and the obligation to update implementations in line with new information and techniques to protect its users and the ecosystem in which it operates.

However, some aspects of [Ethereum](#) in general, and [Enterprise Ethereum](#) specifically, are especially important in an [organizational](#) environment.

[Enterprise Ethereum](#) software development shares with all software development the need to consider security issues and the obligation to update implementations in line with new information and techniques to protect its users and the ecosystem in which it operates.

### 3.1 Callback URL Sanitization

The asynchronous JSON-RPC methods `eea_sendTransactionAsync` and `eea_sendRawTransactionAsync` utilize a URL provided by the user at call time to inform the user of the completion of the asynchronous operation. Attackers can use these URLs to cause the node server to invoke resources present on the nodes private network that the attacker would not normally have access to or to cause the node to spam the callback URL. [Enterprise Ethereum clients](#) need to provide appropriate URL sanitization and restrictions, such as whitelisting and

request throttling, to prevent such vulnerabilities from being exploited in the course of the asynchronous operations execution.

## 3.2 Attacks on Enterprise Ethereum

Modeling attacks against a [node](#) helps identify and prioritize the necessary security countermeasures to implement. Some attack categories to consider include:

- Attacks on unauthenticated [[JSON-RPC](#)] interfaces through malicious JavaScript in the browser using DNS rebinding.
- Eclipse attacks (attacks targeting specific [nodes](#) in a decentralized network) that attempt to exhaust [client](#) network resources or fool its node-discovery protocol.
- Targeted exploitation of consensus bugs in [EVM](#) implementations.
- Malicious code contributions to open-source repositories.
- All varieties of social engineering attacks.

## 3.3 Positive Security Design Patterns

Complex interfaces increase security risk by making user error more likely. For example, entering Ethereum addresses by hand is prone to errors. Therefore, implementations can reduce the risk by providing user-friendly interfaces, ensuring users correctly select an opaque identifier using tools like a contact manager.

**Gas** (defined in the [[Ethereum-Yellow-Paper](#)]) is a virtual pricing mechanism for [transactions](#) and [smart contracts](#) that is implemented by [Ethereum](#) to protect against Denial of Service attacks and resource-consumption attacks by compromised, malfunctioning or malicious [nodes](#). [Enterprise Ethereum](#) provides additional tools to reduce security risks, such as more granular [permissions](#) for actions in a network.

[Permissioning](#) plays some role in mitigating network-level attacks (like the 51% attack), but it is important to carefully consider which risks are of most concern to a [client](#) implementation versus those that are better mitigated by updates to the [Ethereum](#) consensus protocol design.

## 3.4 Handling of Sensitive Data

The implications of private data storage are also important to consider, and motivate several requirements within this Specification.

The long-term persistence of encrypted data on any public platform (such as [Ethereum](#)) exposes it to eventual decryption by brute-force attack, accelerated by the inevitable periodic advances in cryptanalysis. A future shift to post-quantum cryptography is a current concern, but it will likely not be the last advancement in the field. Assuming no encryption scheme endures for eternity, a degree of protection is required to reasonably exceed the lifetime of the data's sensitivity.

Besides user-generated data, a [client](#) is also responsible for managing and protecting private keys. Encrypting private keys with a passphrase or other authentication credential before storage helps protect them from disclosure. It is also important not to disclose sensitive data when recording events to a log file.

### 3.5 Security of Client Implementations

There are several specific functionality areas that are more prone to security issues arising from implementation bugs. The following areas deserve a greater focus during the design and the security assessment of an [Enterprise Ethereum client](#):

- Peer-to-peer protocol implementation
- Object deserialization routines
- [Ethereum Virtual Machine \(EVM\)](#) implementation
- Key pair generation.

The peer-to-peer protocol used for communication among [nodes](#) in [Ethereum](#) is a [client's](#) primary vector for exposure to untrusted input. In any software, the program logic that handles untrusted inputs is the primary focus area for implementing secure data handling.

Object serialization and deserialization is commonly part of the underlying implementation of the P2P protocol, but also a source of complexity that, historically, is prone to security vulnerabilities across many implementations and many programming languages. Selecting a deserializer that offers strict control of data typing can help mitigate the risk.

[EVM](#) implementation correctness is an especially important security consideration for [clients](#). Unless [EVMs](#) behave identically for all possibilities of input, there is a serious risk of a [hard fork](#) caused by an input that elicits the differences in behavior across [clients](#). [EVM](#) implementations are also exposed to denial-of-service attempts by maliciously constructed [smart contracts](#), and the even more serious risk of an exploitable remote-code-execution vulnerability.

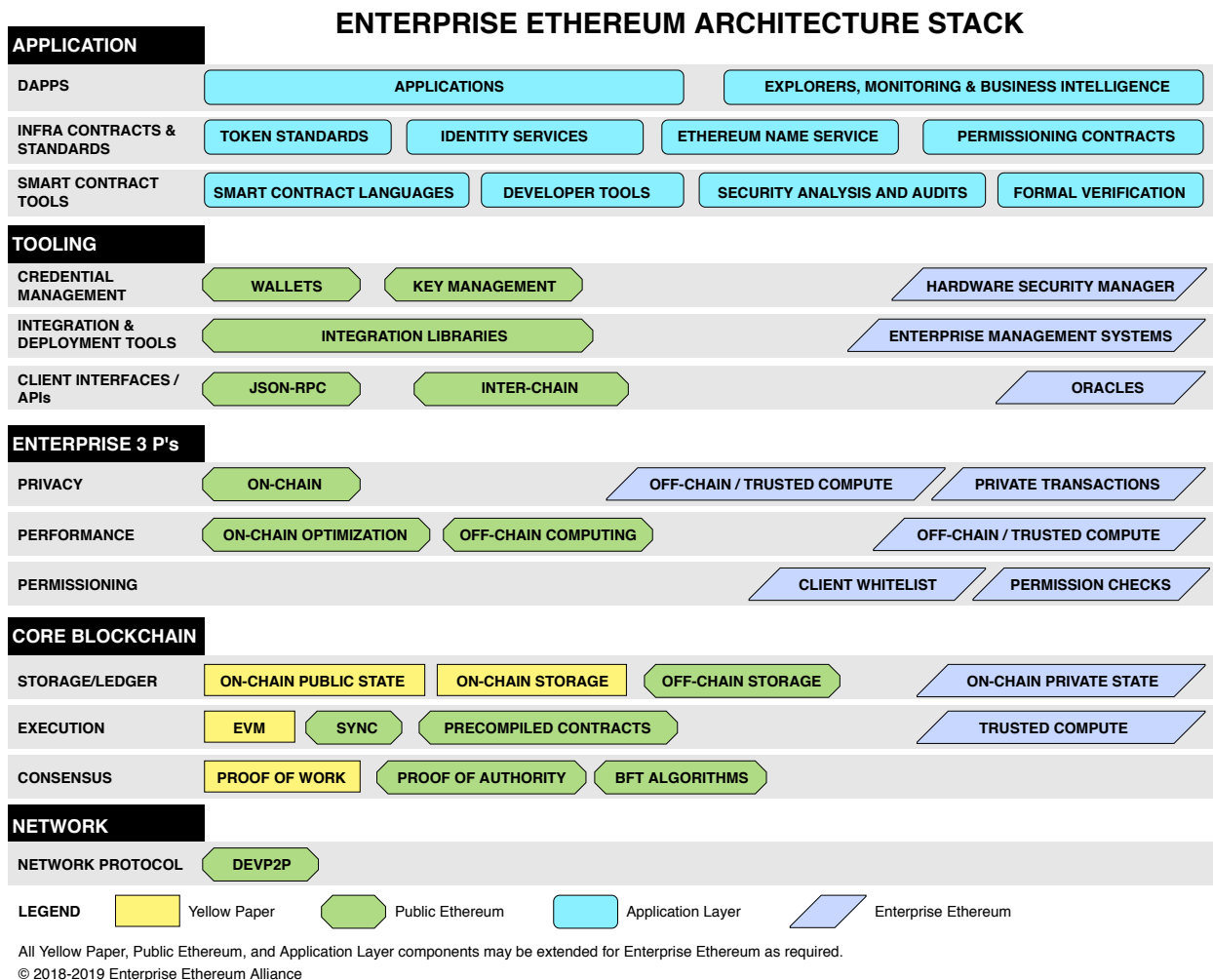
The [Ethereum](#) specification defines many of the technical aspects of public/private key pair format and cryptographic algorithm choice, but a [client](#) implementation is still responsible for properly generating these keys using a well-reviewed cryptographic library. Specifically, a [client](#) implementation needs a properly seeded, cryptographically secure, pseudo-random number generator (PRNG) during the keypair generation step. An insecure PRNG is not generally apparent

by merely observing its outputs, but enables attackers to break the encryption and reveal users' sensitive data.

## 4. Enterprise Ethereum Architecture

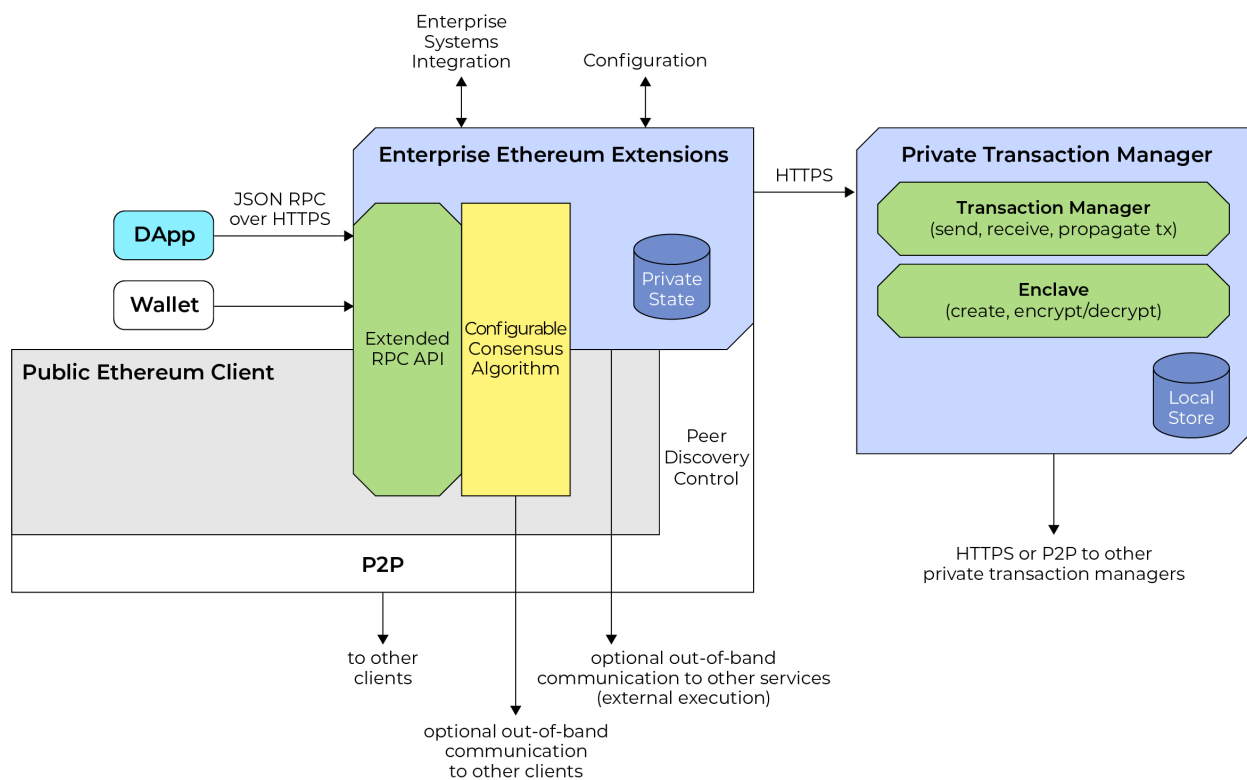
*This section is non-normative.*

The following two diagrams show the relationship between [Enterprise Ethereum](#) components that can be part of any [Enterprise Ethereum client](#) implementation. The first is a stack representation of the architecture showing a library of interfaces, while the second is a more traditional style architecture diagram showing a representative architecture.



*Figure 1 Enterprise Ethereum Architecture Stack*

## ENTERPRISE ETHEREUM HIGH LEVEL ARCHITECTURE



*Figure 2 Representative Enterprise Ethereum High-level Architecture*

The architecture stack for [Enterprise Ethereum](#) consists of five layers:

- Application
- Tooling
- Privacy and Scaling
- Core Blockchain
- Network.

These layers are described in the following sections.

### 5. Application Layer

The Application layer exists, often fully or partially outside of a [client](#), where higher-level services are provided. For example, [Ethereum Name Service](#) (ENS), [node](#) monitors, blockchain state

visualizations and explorers, self-sovereign and other identity schemes, [wallets](#), and any other applications of the ecosystem envisaged.

**Wallets** are software applications used to store an individual's credentials (cryptographic private keys), which are associated with the state of that user's [Ethereum account](#).

[Wallets](#) can interface with [Enterprise Ethereum](#) using the Extended RPC API, as shown in [Figure 2](#). A [wallet](#) can also interface directly with the enclave of a [private transaction manager](#), or interface with [public Ethereum](#).

A **private transaction manager** is a subsystem of an [Enterprise Ethereum](#) system for implementing privacy and [permissioning](#).

## 5.1 DApps Sublayer

Decentralized Applications, or **DApps**, are software applications running on a decentralized peer-to-peer network, often a blockchain. A DApp might include a user interface running on another (centralized or decentralized) system. DApps run on top of [Ethereum](#). DApps running on an [Enterprise Ethereum blockchain](#) can use the extensions to the [Ethereum JSON-RPC API](#) that are defined in this Specification.

Also at the [DApps](#) sublayer are blockchain explorers, tools to monitor the blockchain, and other business intelligence tools.

## 5.2 Infrastructure Contracts and Standards Sublayer

*This section is non-normative.*

Some important tools for managing a blockchain, are built at the Application layer. These components together make up the Infrastructure Contracts and Standards sublayer.

**Permissioning contracts** determine whether [nodes](#) and [accounts](#) can access, or perform specific actions on, an [Enterprise Ethereum blockchain](#), according to the needs of the blockchain. These permissioning contracts can implement Role-based access control (RBAC) [[WP-RBAC](#)] or Attribute-based access control (ABAC) [[WP-ABAC](#)], as well as simpler permissioning models, as described in the Permissioning Management Examples section of the Implementation Guide [[EEA-implementation-guide](#)].

Token standards provide common interfaces and methods along with best practices. These include [[ERC-20](#)], [[ERC-223](#)], [[ERC-621](#)], [[ERC-721](#)], and [[ERC-827](#)].

The **Ethereum Name Service** (ENS) provides a secure and decentralized mapping from simple, human-readable names to [Ethereum](#) addresses for resources both on and off the blockchain.

## 5.3 Smart Contract Tools Sublayer

[Enterprise Ethereum](#) inherits the [smart contract](#) tools used by [public Ethereum](#). These tools include [smart contract languages](#) and associated developer tools such as parsers, compilers, and debuggers, as well as methods used for security analysis and [formal verification](#) of [smart contracts](#).

[Enterprise Ethereum](#) implementations enable use of these tools and methods through implementation of the Execution sublayer, as described in Section § 8.2 [Execution Sublayer](#).

**[P] SMRT-030:** [Enterprise Ethereum clients](#) *MUST* support [smart contracts](#) of at least 24,576 bytes in size.

**[P] SMRT-040:** [Enterprise Ethereum clients](#) *MUST* read and enforce a size limit for [smart contracts](#) from the current [network configuration](#) (for example, from the [genesis block](#)).

**[P] SMRT-050:** If no contract size limit is specified in a [genesis block](#), subsequent [hard fork block](#), or [network configuration](#), [Enterprise Ethereum clients](#) *MUST* enforce a size limit on [smart contracts](#) of 24,576 bytes.

The *genesis block* is the first block of a blockchain.

A *hard fork* is a permanent divergence from the previous version of a blockchain. [Nodes](#) using older [network configuration](#) are no longer able to participate fully in the [Enterprise Ethereum blockchain](#) after the [hard fork block](#).

A *hard fork block* is the block from which a [hard fork](#) occurred.

## 6. Tooling Layer

The Tooling layer contains the APIs used to communicate with [clients](#). The *Ethereum JSON-RPC API*, implemented by [public Ethereum](#), is the primary API to submit [transactions](#) for execution, deploy [smart contracts](#), and to allow [DApps](#) and [wallets](#) to interact with the platform. The [\[JSON-RPC\]](#) remote procedure call protocol and format is used for the JSON-RPC API implementation. Other APIs are allowed, including those intended for inter-blockchain operations and to call external services, such as trusted [oracles](#).

*Integration libraries*, such as [\[web3j\]](#), [\[web3.js\]](#), and [\[Nethereum\]](#), are software libraries used to implement APIs with different language bindings (like the [Ethereum JSON-RPC API](#)) for interacting with [Ethereum nodes](#).

[Enterprise Ethereum](#) implementations can restrict operations based on [permissioning](#) and authentication schemes.

The Tooling layer also provides support for the compilation, and possibly [formal verification](#) of, [smart contracts](#) through the use of parsers and compilers for one or more [smart contract languages](#).

**Smart contract languages** are the programming languages such as [[Solidity](#)] and [[LLL](#)] used to create [smart contracts](#). For each language tools can perform tasks such as compiling to [EVM](#) bytecode, static security checking, or [formal verification](#).

**Formal verification** is the mathematical verification of the logical correctness of a [smart contract](#) designed to run in the [EVM](#).

## 6.1 Credential Management Sublayer

Credentials in the context of [Enterprise Ethereum blockchains](#) refers to an individual's cryptographic private keys, which are associated with that user's [Ethereum account](#). [Enterprise Ethereum clients](#) can choose to offer local handling of [user](#) credentials, such as key management systems and [wallets](#). Such facilities might also be implemented outside the scope of a [client](#).

## 6.2 Integration and Deployment Tools Sublayer

Many software systems integrate with enterprise management systems using common APIs, [libraries](#), and techniques, as shown in [Figure 3](#).

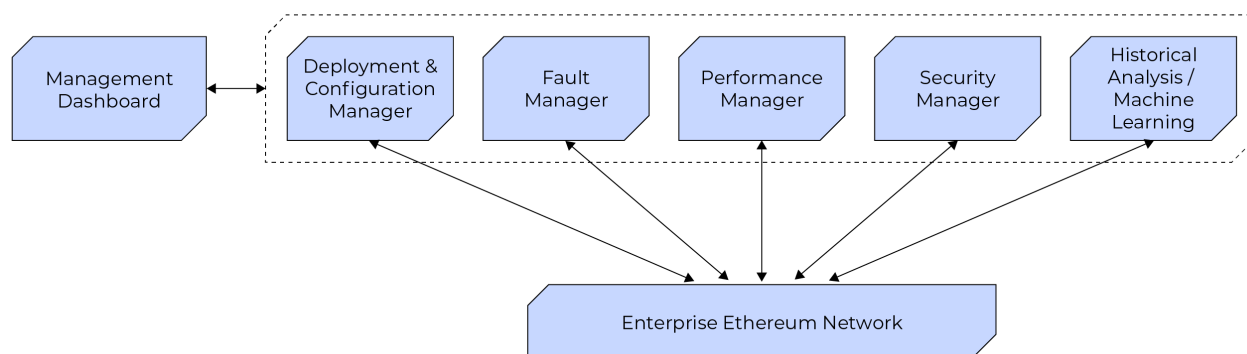


Figure 3 Management Interfaces

As well as deployment and configuration capabilities, [Enterprise Ethereum clients](#) can offer possibilities such as software fault reporting, performance management, security management, integration with other enterprise software, and historical analysis tools.



These are not requirements of this Specification, instead they are optional features to distinguish between different [Enterprise Ethereum clients](#).

## 6.3 Client Interfaces and APIs Sublayer

As part of the Client Interfaces and APIs sublayer, [\[JSON-RPC\]](#) is a stateless, light-weight remote procedure call (RPC) protocol using [\[JSON\]](#) as its data format. The [\[JSON-RPC\]](#) specification defines several data structures and the rules around their processing.

An [Ethereum JSON-RPC API](#) is used to communicate between [DApps](#) and [nodes](#).

### 6.3.1 Compatibility with the Core Ethereum JSON-RPC API

**[P] JRPC-010:** [Enterprise Ethereum clients](#) *MUST* provide support for the following [Ethereum JSON-RPC API](#) methods:

- `net_version`
- `net_peerCount`
- `net_listening`
- `eth_protocolVersion`
- `eth_syncing`
- `eth_coinbase`
- `eth_hashrate`
- `eth_gasPrice`
- `eth_accounts`
- `eth_blockNumber`
- `eth_getBalance`
- `eth_getStorageAt`
- `eth_getTransactionCount`
- `eth_getBlockTransactionCountByHash`
- `eth_getBlockTransactionCountByNumber`
- `eth_getCode`
- `eth_sendRawTransaction`
- `eth_call`
- `eth_estimateGas`

- `eth_getBlockByHash`
- `eth_getBlockByNumber`
- `eth_getTransactionByHash`
- `eth_getTransactionByBlockHashAndIndex`
- `eth_getTransactionByBlockNumberAndIndex`
- `eth_getTransactionReceipt`
- `eth_getUncleByBlockHashAndIndex`
- `eth_getUncleByBlockNumberAndIndex`
- `eth_getLogs`.

[P] **JRPC-007:** [Enterprise Ethereum clients](#) *SHOULD* implement [\[JSON-RPC-API\]](#) methods to be backward compatible with the definitions given in version f4e6248 of the [Ethereum JSON-RPC API](#) reference [\[JSON-RPC-API-vf4e6248\]](#), unless breaking changes were made and widely implemented for the health of the ecosystem. For example, to fix a major security or privacy problem.

[C] **JRPC-015:** [Enterprise Ethereum clients](#) *MUST* provide the capability to accept and respond to JSON-RPC method calls over a websocket interface.

[C] **JRPC-040:** [Enterprise Ethereum clients](#) *MUST* provide an implementation of the `debug_traceTransaction` method [\[debug-traceTransaction\]](#) from the Go Ethereum Management API.

[C] **JRPC-050:** [Enterprise Ethereum clients](#) *MUST* implement the [\[JSON-RPC-PUB-SUB\]](#) API.

[P] **JRPC-070:** [Enterprise Ethereum clients](#) implementing additional nonstandard subscription types for the [\[JSON-RPC-PUB-SUB\]](#) API *MUST* prefix their subscription type names with a namespace prefix other than `eea_`.

### 6.3.2 Extensions to the JSON-RPC API

[P] **JRPC-080:** The [\[JSON-RPC\]](#) method name prefix `eea_` *MUST* be reserved for future use for RPC methods specific to the EEA.

[P] **JRPC-020:** [Enterprise Ethereum clients](#) *MUST* provide one of the following sets of extensions to create [private transaction](#) types defined in Section [§ 7.1.3 Private Transactions](#):

- `eea_sendTransactionAsync` and `eea_sendTransaction`, or
- `eea_sendRawTransactionAsync` and `eea_sendRawTransaction`.

[P] **JRPC-030:** The `eea_sendTransactionAsync`, `eea_sendTransaction`, `eea_sendRawTransactionAsync`, and `eea_sendRawTransaction` methods *MUST* respond with a [JSON-RPC] error response when an unimplemented [private transaction](#) type is requested. The error response *MUST* have the `code` `-50100` and the `message` `Unimplemented private transaction type`.

#### Example response

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "error": {
    "code": -50100,
    "message": "Unimplemented private transaction type"
  }
}
```

#### NOTE

As in the [public Ethereum \[JSON-RPC-API\]](#), the two key datatypes for these `eea_send*Transaction*` calls, which are passed hex encoded, are unformatted data byte arrays (DATA) and quantities (QUANTITY). When encoding unformatted data, encode as hex, prefix with "0x", and use two hex digits per byte. When encoding quantities (integers and numbers), encode as hex and prefix with "0x". When encoding the `privateFrom`, `privateFor`, and `privacyGroupId` DATA fields, encode them as base64.

#### 6.3.2.1 `eea_sendTransactionAsync`

*This section is experimental.*

A call to `eea_sendTransactionAsync` creates a [private transaction](#), signs it, submits it to the [transaction](#) pool, and returns immediately.

Using this method allows sending many [transactions](#) without waiting for recipient confirmation.

#### Parameters

The [transaction](#) object for this call contains:

- `from` DATA, 20 bytes – The address of the [account](#) sending the [transaction](#).
- `to` DATA, 20 bytes – The address of the [account](#) receiving the [transaction](#).
- `gas` QUANTITY – Optional. The [gas](#), as an integer, provided for the [transaction](#).

- **gasPrice** QUANTITY – Optional. The [gas](#) price, as an integer.
- **value** QUANTITY – Optional. The value, as an integer, if present must be set to 0.
- **data** DATA – [Transaction](#) data (compiled [smart contract](#) code or encoded method data).
- **nonce** QUANTITY – Optional. A nonce value, as an integer. This allows you to overwrite your own pending [transactions](#) that use the same nonce.
- **privateFrom** DATA, 32 bytes – Optional. The public key of the sender of this [private transaction](#). If this parameter is not supplied, the node could supply a default for **privateFrom**. If this parameter is not supplied and the node is unable to supply a default, the transaction fails.
- **privateFor** DATA – An array of the public keys of the intended recipients of this [private transaction](#). Mutually exclusive with the **privacyGroupId** parameter. If both the **privateFor** and **privacyGroupId** parameters are provided, an error response is generated.
- **privacyGroupId** DATA, 32 bytes – The privacy group identifier for the group of intended recipients of this [private transaction](#). If a [client](#) does not support this parameter it should return a "PrivacyGroupId not supported" error response. Mutually exclusive with the **privateFor** parameter. If both the **privateFor** and **privacyGroupId** parameters are provided, an error response is generated.
- **restriction** STRING – If **restricted**, the [transaction](#) is a [restricted private transaction](#). If **unrestricted**, the [transaction](#) is an [unrestricted private transaction](#). For more information, see Section [§ 7.1.3 Private Transactions](#).
- **callbackUrl** STRING – The URL to post the results of the [transaction](#) to.

## Callback Body

The callback object for this call contains:

- **txHash** DATA, 32 bytes – The [transaction](#) hash (if successful).
- **txIndex** QUANTITY – The index position, as an integer, of the [transaction](#) in the block.
- **blockHash** DATA, 32 Bytes – The hash of the block this [transaction](#) was in.
- **blockNumber** QUANTITY – The number of the block, as an integer, this [transaction](#) was in.
- **from** DATA, 20 Bytes – The public key of the sender of this [private transaction](#).
- **to** DATA, 20 Bytes – The [account](#) address of the receiver. **null** if a [contract](#) creation [transaction](#).
- **cumulativeGasUsed** QUANTITY – The total amount of [gas](#) used when this [transaction](#) was executed in the block.
- **gasUsed** QUANTITY – The amount of [gas](#) used by this specific [transaction](#).

- **contractAddress** DATA, 20 Bytes – The [contract](#) address created, if a [contract](#) creation [transaction](#), otherwise **null**.
- **logs** Array – An array of log objects generated by this [transaction](#).
- **logsBloom** DATA, 256 Bytes – A bloom filter for light [clients](#) to quickly retrieve related logs.
- **error** STRING – Optional. Includes an error message describing what went wrong.
- **id** DATA – Optional. The ID of the request corresponding to this [transaction](#), as provided in the initial [\[JSON-RPC\]](#) call.

Also returned is either:

- **root** DATA, 32 bytes – The post-transaction stateroot (pre-Byzantium).
- **status** QUANTITY – The return status, either 1 (success) or 0 (failure).

## Request Format

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eea_sendTransactionAsync","params":[{"from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
"to": "0xd46e8dd67c5d32be8058bb8eb970870f072445675",
"gas": "0x76c0",
"gasPrice": "0x9184e72a000",
"data":"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb
"privateFrom": "negmDcN2P40DpqN/6WkJ02zT/0w0bjhGpkZ8UP6vARk=",
"privateFor": ["g59BmTeJIn7HIcnq8VQWgyh/pDbvbt2eyP0Ii60aDDw="],
"callbackUrl": "http://myserver/id=1",
"restriction": "restricted"}],
"id":1}'
```

Or alternatively, when a privacyGroupId is provided instead of privateFor

```
"privacyGroupId": "Vbj70zF+G2V/8XoyZzwqawfcQ+r9BkXoLQ0qkQideys=",
```

## Response Format

```
{
  "id":1,
  "jsonrpc": "2.0"
}
```

## Callback Format

```
{
  "txHash":
    "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
  "txIndex": "0x1", // 1
  "blockNumber": "0xb", // 11
  "blockHash": "0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed"
  "cumulativeGasUsed": "0x33bc", // 13244
  "gasUsed": "0x4dc", // 1244
  "contractAddress": "0xb60e8dd61c5d32be8058bb8eb970870f07233155", // or nu
  "logs": "[{
    // logs as returned by getFilterLogs, etc.
  }, ...]",
  "logsBloom": "0x00...0", // 256 byte bloom filter
  "status": "0x1"
}
```

### 6.3.2.2 *eea\_sendTransaction*

Creates a [private transaction](#), signs it, generates the [transaction](#) hash and submits it to the [transaction](#) pool, and returns the [transaction](#) hash.

#### Parameters

The [transaction](#) object containing:

- **from** DATA, 20 bytes – The address of the [account](#) sending the [transaction](#).
- **to** DATA, 20 bytes – Optional when creating a new [contract](#). The address of the [account](#) receiving the [transaction](#).
- **gas** QUANTITY – Optional. The [gas](#), as an integer, provided for the [transaction](#).
- **gasPrice** QUANTITY – Optional. The [gas](#) price, as an integer.
- **value** QUANTITY – Optional. The value, as an integer, if present must be set to 0.
- **data** DATA – [Transaction](#) data (compiled [smart contract](#) code or encoded method data).
- **nonce** QUANTITY – Optional. A nonce value, as an integer. This allows you to overwrite your own pending [transactions](#) that use the same nonce.
- **privateFrom** DATA, 32 bytes – Optional. The public key of the sender of this [private transaction](#). If this parameter is not supplied, the node could supply a default for **privateFrom**. If this parameter is not supplied and the node is unable to supply a default, the transaction fails.
- **privateFor** DATA – An array of the public keys of the intended recipients of this [private transaction](#). Mutually exclusive with the **privacyGroupId** parameter. If both **privateFor**

and `privacyGroupId` parameters are provided, an error response is generated.

- `privacyGroupId` DATA, 32 bytes – The privacy group identifier for the group of intended recipients of this [private transaction](#). If a [client](#) does not support this parameter it should return a "PrivacyGroupId not supported" error response. Mutually exclusive with the `privateFor` parameter. If both `privateFor` and `privacyGroupId` parameters are provided, an error response is generated.
- `restriction` STRING – If `restricted`, the [transaction](#) is a [restricted private transaction](#). If `unrestricted`, the [transaction](#) is an [unrestricted private transaction](#). For more information, see Section [§ 7.1.3 Private Transactions](#).

## Returns

DATA, 32 Bytes – The [transaction](#) hash, or the zero hash if the [transaction](#) is not yet available.

If creating a [contract](#), use `eth_getTransactionReceipt` to retrieve the [contract](#) address after the [transaction](#) is [finalized](#).

## Request Format

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eea_sendTransaction","params": [{
  "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
  "to": "0xd46e8dd67c5d32be8058bb8eb970870f072445675",
  "gas": "0x76c0",
  "gasPrice": "0x9184e72a000",
  "data":
    "0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb9708
    "privateFrom": "negmDcN2P40Dpqn/6WkJ02zT/0w0bjhGpkZ8UP6vARK=",
    "privateFor": ["g59BmTeJIn7HICnq8VQWgyh/pDbvbt2eyP0Ii60aDDw="],
    "restriction": "restricted"}],
  "id":1}'
```

Or alternatively, when a `privacyGroupId` is provided instead of `privateFor`

```
"privacyGroupId": "Vbj70zF+G2V/8XoyZzwqawfcQ+r9BkXoLQ0qkQideys=",
```

## Response Format

```
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527
}
```

### 6.3.2.3 *eea\_sendRawTransaction*

Creates a [private transaction](#), which has already been signed, generates the [transaction](#) hash and submits it to the [transaction](#) pool, and returns the [transaction](#) hash.

The signed [transaction](#) passed as an input parameter is expected to include the [privateFrom](#), [privateFor](#), [privacyGroupId](#), and [restriction](#) fields, as specified in the Parameters section of [§ 6.3.2.2 eea\\_sendTransaction](#).

## Parameters

The [transaction](#) object containing:

- **data** DATA – The signed [transaction](#) data.

```
params: ["0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058"]
```

## Returns

DATA, 32 Bytes – The [transaction](#) hash, or the zero hash if the [transaction](#) is not yet available.

If creating a [contract](#), use [eth\\_getTransactionReceipt](#) to retrieve the [contract](#) address after the [transaction](#) is [finalized](#).

## Request Format

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eea_sendRawTransaction","params": [{see above
"id":1}]'
```

## Response Format

```
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527"
}
```

### 6.3.2.4 eea\_sendRawTransactionAsync

*This section is experimental.*

A call to [eea\\_sendRawTransactionAsync](#) creates a [private transaction](#), which has already been signed, submits it to the [transaction](#) pool, and returns immediately.

Using this method allows sending many [transactions](#) without waiting for recipient confirmation.



The signed [transaction](#) passed as an input parameter is expected to include the **privateFrom**, **privateFor**, **privacyGroupId**, and **restriction** fields, as specified in the Parameters section of § 6.3.2.1 [eea\\_sendTransactionAsync](#). It is also expected to include the **callbackUrl** field.

## Parameters

The [transaction](#) object containing:

- **data** DATA – The signed [transaction](#) data.

```
params: ["0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058"]
```

## Callback Body

The callback object for this call contains:

- **txHash** DATA, 32 bytes – The [transaction](#) hash (if successful).
- **txIndex** QUANTITY – The index position, as an integer, of the [transaction](#) in the block.
- **blockHash** DATA, 32 Bytes – The hash of the block this [transaction](#) was in.
- **blockNumber** QUANTITY – The number of the block, as an integer, this [transaction](#) was in.
- **from** DATA, 20 Bytes – The public key of the sender of this [private transaction](#).
- **to** DATA, 20 Bytes – The address of the [account](#) receiving this [transaction](#). **null** if a [contract creation transaction](#).
- **cumulativeGasUsed** QUANTITY – The total amount of [gas](#) used when this [transaction](#) was executed in the block.
- **gasUsed** QUANTITY – The amount of [gas](#) used by this specific [transaction](#).
- **contractAddress** DATA, 20 Bytes – The [contract](#) address created, if a [contract creation transaction](#), otherwise **null**.
- **logs** Array – An array of log objects generated by this [transaction](#).
- **logsBloom** DATA, 256 Bytes – A bloom filter for light [clients](#) to quickly retrieve related logs.
- **error** STRING – Optional. Includes an error message describing what went wrong.
- **id** DATA – Optional. The ID of the request corresponding to this [transaction](#), as provided in the initial [\[JSON-RPC\]](#) call.

Also returned is either:

- **root** DATA, 32 bytes – The post-transaction stateroot (pre-Byzantium).
- **status** QUANTITY – The return status, either 1 (success) or 0 (failure).

## Request Format

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eea_sendRawTransactionAsync","params": [{see
"id":1}]'
```

## Response Format

```
{
  "id":1,
  "jsonrpc": "2.0"
}
```

## Callback Format

```
{
  "txHash":
    "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
  "txIndex": "0x1", // 1
  "blockNumber": "0xb", // 11
  "blockHash": "0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed
  "cumulativeGasUsed": "0x33bc", // 13244
  "gasUsed": "0x4dc", // 1244
  "contractAddress": "0xb60e8dd61c5d32be8058bb8eb970870f07233155", // or nu
  "logs": "[{
    // logs as returned by getFilterLogs, etc.
  }, ...]",
  "logsBloom": "0x00...0", // 256 byte bloom filter
  "status": "0x1"
}
```

### 6.3.3 Permissioning Smart Contract

This section presents [smart contract](#) interfaces providing the necessary information for [Enterprise Ethereum clients](#) to enforce [permissioning](#) models in an [interoperable](#) manner. This includes both [node-](#) and [account-permissioning](#) interfaces.

It is based on a chain deployment architecture where [permissioning](#) is split into [permissioning management](#), handled by a [permissioning contract](#) on the [Enterprise Ethereum blockchain](#), and [permissioning enforcement](#), handled by the [Enterprise Ethereum client](#) based on information provided by the [permissioning contract](#).

#### 6.3.3.1 Permissioning enforcement

For information necessary to enforce the permissioning requirements of an [Enterprise Ethereum blockchain](#), [Enterprise Ethereum clients](#) call specific functions in the [permissioning contracts](#). These are common functions for all [clients](#) on the [Enterprise Ethereum blockchain](#) to use. These functions include:

#### **[connectionAllowed](#)**

Determines whether to permit a connection with another [node](#).

#### **[transactionAllowed](#)**

Determines whether to accept a [transaction](#) received from a given [Ethereum account](#).

A [client](#) is not necessarily able to update the [permissioning](#) scheme, nor does it automatically have any knowledge of its implementation.

The [node-](#) and [account-permissioning](#) interfaces emit a **[permissionsUpdated](#)** event when the underlying rules are changed. [Clients](#) register for these events that signal when to re-assess any [permissions](#) that were granted, and when to re-assess any [permission](#) check results that were cached.

The event contains **[addsRestrictions](#)** and **[addsPermissions](#)** Boolean flags. If either flag is set to **true**, any previous [connectionAllowed](#) or [transactionAllowed](#) call could now result in a different outcome, as the previously checked [permissions](#) have changed. If **[addsRestrictions](#)** is set to **true**, this indicates that one or more previous [connectionAllowed](#) or [transactionAllowed](#) calls that returned **true** will now return **false**, and analogously if **[addsPermissions](#)** is **true** at least one [connectionAllowed](#) or [transactionAllowed](#) call that returned **false** will now return **true**.

### 6.3.3.2 Permissioning management

These [smart contract](#) functions provide the ability to configure and manage the [permissioning](#) model in use. These include the bulk of the constructs used to organize [permissions](#), processes to adjust [permissions](#), administration of the [permissioning](#) mechanism, and enforcing any regulatory requirements.

The definition of these **[permissioning management](#)** functions depends on the [permissioning](#) model of the specific [Enterprise Ethereum blockchain](#). It is outside the scope of this Specification, but crucial to the operation of the system.

[Enterprise Ethereum blockchain](#) operators can choose any [permissioning](#) model that suits their needs.

Implementations of the [permissioning contracts](#) (both enforcement and management functions) are provided on the [Enterprise Ethereum blockchain](#) by the blockchain operator. The implementation

of [permissioning enforcement](#) functions, such as [connectionAllowed](#), is part of the [permissioning contract](#).

When a management function is called that updates the [permissioning](#) model, the [node](#) or [account smart contract](#) interfaces emit a [permissionsUpdated](#) event based on the [permissions](#) change.

### 6.3.3.3 Node Permissioning

**Node permissioning** restricts the peer connections that can be established with other [nodes](#) in the [Enterprise Ethereum blockchain](#). This helps to prevent interference and abuse by external parties and can establish a trusted whitelist of [nodes](#).

**[P] PERM-200:** [Enterprise Ethereum clients](#) *MUST* call the [connectionAllowed](#) function, as specified in Section § 6.3.3.3.1 [Node Permissioning Functions](#), or if it implements [PERM-220](#) and [PERM-230](#), *MAY* use cached information to determine whether a connection with another [node](#) is permitted, and any restrictions to be placed on that connection.

The [connectionAllowed](#) function returns a [bytes32](#) type, which is interpreted as a bitmask with each bit representing a specific [permission](#) for the connection.

**[P] PERM-210:** When checking the response to [connectionAllowed](#), if any unknown permissioning bits are found to be zero, [Enterprise Ethereum clients](#) *MUST* reject the connection.

**[P] PERM-220:** On receipt of a [NodePermissionsUpdated](#) event containing an [addsRestrictions](#) property with the value [true](#), [Enterprise Ethereum clients](#) *MUST* close any network connections that are no longer permitted, and impose newly added restrictions on any network connections that have had restrictions added.

**[P] PERM-230:** On receipt of a [NodePermissionsUpdated](#) event containing an [addsPermissions](#) property with the value [true](#), [Enterprise Ethereum clients](#) *MUST* check whether existing network connections have had their restrictions lifted and allow future actions that are now permitted.

#### 6.3.3.3.1 NODE PERMISSIONING FUNCTIONS

The [node](#) connection rules support both the IPv4 and IPv6 protocol versions. IPv6 addresses are represented using their logical byte value with big endian byte ordering. IPv4 addresses are specified in the IPv4 reserved space within the IPv6 address space, which is found at [0000:0000:0000:0000:0000:ffff:](#), and can be assembled by taking the logical byte value of the IPv4 address with big endian byte ordering, and prefixing it with 80 bits of zeros followed by 16 bits of ones.

The **connectionAllowed** function is found at the address given by the **nodePermissionContract** parameter in the [network configuration](#). It implements the following interface, including the **NodePermissionsUpdated** [permissionsUpdated](#) event:

## Interface

```
[
  {
    "name": "connectionAllowed",
    "stateMutability": "view",
    "type": "function",
    "inputs": [
      {
        "name": "sourceEnodeHigh",
        "type": "bytes32"
      },
      {
        "name": "sourceEnodeLow",
        "type": "bytes32"
      },
      {
        "name": "sourceIp",
        "type": "bytes16"
      },
      {
        "name": "sourcePort",
        "type": "uint16"
      },
      {
        "name": "destinationEnodeHigh",
        "type": "bytes32"
      },
      {
        "name": "destinationEnodeLow",
        "type": "bytes32"
      },
      {
        "name": "destinationIp",
        "type": "bytes16"
      },
      {
        "name": "destinationPort",
        "type": "uint16"
      }
    ],
    "outputs": [
      {
        "name": "result",
        "type": "bytes32"
      }
    ]
  }
],
```

```

{
  "type": "event",
  "name": "NodePermissionsUpdated",
  "inputs": [
    {
      "name": "addsRestrictions",
      "type": "bool",
      "indexed": false
    },
    {
      "name": "addsPermissions",
      "type": "bool",
      "indexed": false
    }
  ]
}
]

```

## Arguments

- **sourceNodeHigh**: The high (first) 32 bytes of the enode address of the [node](#) initializing the connection.
- **sourceNodeLow**: The low (last) 32 bytes of the enode address of the [node](#) initiating the connection.
- **sourceIp**: The IP address of the [node](#) initiating the connection. If the address is IPv4, it should be prefixed by 80 bits of zeros and 16 bits of ones, bitmasking it such that it fits the IPv4 reserved space in IPv6. For example, **::ffff:127.0.0.1**.
- **sourceNodePort**: The peer-to-peer listening port of the [node](#) initiating the connection.
- **destinationNodeHigh**: The high (first) 32 bytes of the enode address of the [node](#) receiving the connection.
- **destinationNodeLow**: The low (last) 32 bytes of the enode address of the [node](#) receiving the connection.
- **destinationIp**: The IP address of the [node](#) receiving the connection. If the address is IPv4, it should be prefixed by 80 bits of zeros and 16 bits of ones, bitmasking it such that it fits the IPv4 reserved space in IPv6. For example, **::ffff:127.0.0.1**.
- **destinationNodePort**: The peer-to-peer listening port of the [node](#) receiving the connection.
- **result**: A bitmask of the [permissions](#) granted for this connection.
- **addsRestrictions**: If the rules change that caused the **NodePermissionsUpdated** event to be emitted involves further restricting existing [permissions](#), this will be **true**. Otherwise it will be **false**.

- **addsPermissions**: If the rules change that caused the NodePermissionsUpdated event to be emitted involves granting new permissions, this will be **true**. Otherwise it will be **false**.

#### 6.3.3.3.2 NODE PERMISSIONS

While the core premise of node permissioning is whether a connection is allowed to occur or not, there are additional restrictions that can be imposed on a connection between two nodes based on the permitted behavior of the nodes.

The various permissions that can be granted to a connection are represented by bits being set in the bitmask response from connectionAllowed. Where bits are unset, the client restricts the behavior of the remote node according to the unset bits.

The remaining bits in the response are normally set to one. If any of the remaining bits are zero, an unknown permission restriction was placed on the connection and the connection will be denied. These unknown zeros are likely to represent permissions defined in future versions of this specification. Where they cannot be interpreted by a client the connection is rejected.

#### Connection Permitted

Permission Bit Index: 0

The connection is allowed to be established.

#### 6.3.3.3.3 CLIENT IMPLEMENTATION

A client connecting to a chain that maintains a permissioning contract finds the address of the contract in the network configuration. When a peer connection request is received, or a new connection request initiated, the permissioning contract is queried to assess whether the connection is permitted. If permitted, the connection is established and when the node is queried for peer discovery, this connection can be advertised as an available peer. If not permitted, the connection is either refused or not attempted, and the peer excluded from any responses to peer discovery requests.

During client startup and initialization the client will begin at a bootnode and initially have a global state that is out of sync. Until the client reaches a trustworthy head it is unable to reach a current version of the node permissioning that correctly represents the current blockchain's state.

#### 6.3.3.3.4 CHAIN INITIALIZATION



At the [genesis block](#) an initial [permissioning contract](#) will normally be included in block 0, configured so the initial [nodes](#) are able to establish connections to each other.

#### 6.3.3.4 Account Permissioning

*Account permissioning* controls which [accounts](#) are able to send [transactions](#) and the type of [transactions](#) permitted.

[P] **PERM-240:** When validating or mining a block, [Enterprise Ethereum clients](#) *MUST* call the [transactionAllowed](#) function, as specified in Section § 6.3.3.4.1 [Account Permissioning Function](#), with worldstate as at the block's parent, or if it implements **[PERM-250](#req-perm-250)]** and **[PERM-260](#req-perm-260)]** *MAY* use cached information, to determine if a [transaction](#) is permitted in a block.

[P] **PERM-250:** On receipt of an [AccountPermissionsUpdated](#) event containing an [addsRestrictions](#) property with the value **true**, [Enterprise Ethereum clients](#) *MUST* purge all cached results from previous calls to [transactionAllowed](#) where the result returned was **true**.

[P] **PERM-260:** On receipt of an [AccountPermissionsUpdated](#) event containing an [addsPermissions](#) property with the value **true**, [Enterprise Ethereum clients](#) *MUST* purge all cached results from previous calls to [transactionAllowed](#) where the result returned was **false**.

##### 6.3.3.4.1 ACCOUNT PERMISSIONING FUNCTION

The [transactionAllowed](#) function is found at the address given by the [transactionPermissionContract](#) parameter in the [network configuration](#). It implements the following interface, including the [AccountPermissionsUpdated](#) event:

## Interface

```
[
  {
    "name": "transactionAllowed",
    "stateMutability": "view",
    "type": "function",
    "inputs": [
      {
        "name": "sender",
        "type": "address"
      },
      {
        "name": "target",
        "type": "address"
      },
      {
        "name": "value",
        "type": "uint256"
      },
      {
        "name": "gasPrice",
        "type": "uint256"
      },
      {
        "name": "gasLimit",
        "type": "uint256"
      },
      {
        "name": "payload",
        "type": "bytes"
      }
    ],
    "outputs": [
      {
        "name": "result",
        "type": "bool"
      }
    ]
  },
  {
    "type": "event",
    "name": "AccountPermissionsUpdated",
    "inputs": [
      {
        "name": "addsRestrictions",
        "type": "bool",
        "indexed": false
      }
    ]
  }
]
```

```

    },
    {
      "name": "addsPermissions",
      "type": "bool",
      "indexed": false
    }
  ]
}
]

```

## Arguments

- **sender**: The address of the [account](#) that created this [transaction](#).
- **target**: The address of the [account](#) or [contract](#) that this [transaction](#) is directed at. For a creation [contract](#) where there is no target, this should be zero filled to represent the **null** address.
- **value**: The eth value being transferred in this [transaction](#).
- **gasPrice**: The [gas](#) price included in this [transaction](#)
- **gasLimit**: The [gas](#) limit in this [transaction](#).
- **payload**: The payload in this [transaction](#). Either empty if a simple value [transaction](#), the calling payload if executing a [contract](#), or the [EVM](#) code to be deployed for a [contract](#) creation.
- **result**: A Boolean value representing whether the [transaction](#) should be allowed and considered valid.
- **addsRestrictions**: If the rules change that caused the [AccountPermissionsUpdated](#) event to be emitted involves further restricting existing [permissions](#), this will be **true**.
- **addsPermissions**: If the rules change that caused the [AccountPermissionsUpdated](#) event to be emitted grants new [permissions](#), this will be **true**.

### 6.3.3.4.2 CLIENT IMPLEMENTATION

A [client](#) connecting to a chain that maintains a [permissioning contract](#) can find address of the [transactionAllowed](#) function in the [transactionPermissionContract](#) parameter of the [network configuration](#).

When mining new blocks the [node](#) checks the validity of [transactions](#) using the appropriate [permissioning contract](#) with the state at the block's parent. If not permitted, the [transaction](#) is discarded. If permitted, the [transaction](#) is included in the new block and the block dispatched to other [nodes](#).

When receiving a block the [node](#) checks each included [transaction](#) using the [permissioning contract](#) with the state at the block's parent. If any [transactions](#) in the new block are not permitted, the block is considered invalid and discarded. If all [transactions](#) are permitted, the block passes the [permissioning](#) validation check and is then subject to any other validity assessments the [client](#) might usually perform.

Depending on the use case of a [client](#), the implementation can also check validity of [transactions](#) submitted through RPC methods or received through peer-to-peer communication. For such validation, it is expected that the [contracts](#) are used with the state represented at the current head.

Reading of a [contract](#) is an unrestricted operation.

#### 6.3.3.4.3 CONTRACT IMPLEMENTATION

When a [transaction](#) is checked by the [contract](#) it can be assessed by any of the fields provided to restrict operations, such as transferring value between [accounts](#), rate limiting spend or receipt of value, restricting the ability to execute code at an address, limiting [gas](#) expenditure or enforcing a minimum expenditure, or restricting the scope of a created [contract](#).

When checking the execution of code at an address, it can be useful to be aware of the [EXTCODEHASH](#) [EVM](#) operation, which allows for checking whether there is code present to be executed at the address that received the request.

For restricting the scope of created [contracts](#) it might be necessary to do static code analysis of the [EVM](#) bytecode payload for properties that are not allowed. For example, restricting creation of [contracts](#) that employ the create contract opcode.

#### 6.3.3.4.4 CHAIN INITIALIZATION

At the [genesis block](#) the [permissioning contract](#) function is included in block 0, configured so initial [accounts](#) can perform required value [transactions](#), a predetermined set of [accounts](#) can invoke the [contracts](#) defined in the genesis file, and if desired, a predetermined set of [accounts](#) can create new [contracts](#).

### 6.3.4 Inter-chain

*This section is non-normative.*

With the rapid expansion in the number of different blockchains and ledgers, ***inter-chain mediators*** allow interaction between these blockchains. Like other solutions that provide privacy

and scalability, inter-chain mediators can be built in Layer 2, such as using [public Ethereum](#) to anchor state as needed for tracking and checkpoints.

### 6.3.5 Oracles

In many situations, [smart contracts](#) need to interact with real-world information to operate. An **oracle** is a service external to either [public Ethereum](#) or an [Enterprise Ethereum client](#) that is trusted by the creators of [smart contracts](#) and is called to provide information, such as a current exchange rate, or the result of a mathematical calculation. Oracles are a secure bridge between [smart contracts](#) and real-world information sources.

[C] ORCL-010: [Enterprise Ethereum clients](#) *SHOULD* provide the ability to securely interact with [oracles](#) to send and receive external off-chain information.

## 7. Enterprise 3 P's Layer

Along with [permissioning](#), the "3 Ps" of [Enterprise Ethereum](#) include **privacy** and **performance**. This layer describes the extensions in [Enterprise Ethereum](#) that support these requirements.

Privacy and performance solutions are broadly categorized into:

- **Layer 1** solutions, which are implemented at the base level protocol layer using techniques such as [sharding](#) and easy parallelizability [\[EIP-648\]](#).
- **Layer 2** solutions, which do not require changes to the base level protocol layer. They are implemented at the application protocol layer, for example using [\[Plasma\]](#), [\[state-channels\]](#), and [Off-Chain Trusted Computing](#) mechanisms.

### 7.1 Privacy Sublayer

Many use cases for [Enterprise Ethereum blockchains](#) have to comply with regulations related to privacy. For example, banks in the European Union are required to comply with the European Union revised Payment Services Directive [\[PSD2\]](#) when providing payment services, and the General Data Protection Regulation [\[GDPR\]](#) when storing personal data regarding individuals.

[Enterprise Ethereum clients](#) support privacy with various techniques including [private transactions](#) and enabling an [Enterprise Ethereum blockchain](#) to permit anonymous participants. They can also support privacy-enhanced [Off-Chain Trusted Computing](#).

Various new privacy mechanisms are being explored as extensions to [public Ethereum](#), such as **zero-knowledge proofs** [\[ZKP\]](#), a cryptographic technique where one party (the prover) can prove to another party (the verifier) that the prover knows a value  $x$ , without conveying any information

apart from the fact that the prover knows the value. [\[ZK-STARKS\]](#) is an example of a zero-knowledge proof method.

- A **transaction** is a core component of most blockchains, including [Public Ethereum](#) as well as [Enterprise Ethereum](#). It is a request to execute operations that change the state of one or more [accounts](#). [Nodes](#) processing [transactions](#) is the fundamental basis of adding blocks to the chain.
- A **private transaction** is a [transaction](#) where some information about the [transaction](#), such as the [payload data](#), or the sender or the recipient, is only available to the subset of parties privy to that [transaction](#).

[Enterprise Ethereum clients](#) support at least one form of [private transaction](#), as outlined in Section § 7.1.3 [Private Transactions](#). [Private transactions](#) can be realized in various ways, controlling which [nodes](#) see which [private transactions](#) or [transaction](#) data.

[Enterprise Ethereum](#) implementations can also support off-chain [Trusted Computing](#), enabling privacy during code execution.

### 7.1.1 On-chain

*This section is non-normative.*

Various on-chain techniques can improve the security and privacy capabilities of [Enterprise Ethereum blockchains](#).

#### NOTE: On-chain Security Techniques

Future on-chain security techniques could include techniques such as [\[ZK-STARKS\]](#), range proofs, or ring signatures.

### 7.1.2 Off-chain (Trusted Computing)

*This section is non-normative.*

**Off-chain trusted computing** uses a privacy-enhanced system to handle some of the computation requested by a [transactions](#). Such systems can be hardware-based, software-based, or a hybrid, depending on the use case.

The EEA has developed Trusted Computing APIs for Ethereum-compatible [trusted computing \[EEA-OC\]](#), and requirement [EXEC-050](#) enables [Enterprise Ethereum clients](#) to use them for improved privacy.

### 7.1.3 Private Transactions

[private transactions](#) specify their preferred type at runtime with the **restriction** parameter on their [\[JSON-RPC-API\]](#) calls. The two defined [private transaction](#) types are:

- **Restricted private transactions**, where [payload data](#) is transmitted to and readable only by the parties to the [transaction](#).
- **Unrestricted private transactions**, where encrypted [payload data](#) is transmitted to all [nodes](#) in the [Enterprise Ethereum blockchain](#), but readable only by the parties to the [transaction](#).

**[P] PRIV-010:** [Enterprise Ethereum clients](#) *MUST* support one of [restricted private transactions](#) or [unrestricted private transactions](#).

[Transaction](#) information consists of two parts:

- **Metadata**, which is the set of data that describes and gives information about the [payload data](#) in a [transaction](#). Metadata is the *envelope* information necessary to execute a [transaction](#).
- **Payload data**, which is the content of the data field of a [transaction](#), usually obfuscated in [private transactions](#). Payload data is separate from the [metadata](#) in a [transaction](#).

If implementing [restricted private transactions](#):

- **[P] PRIV-020:** [Enterprise Ethereum clients](#) *MUST* encrypt [payload data](#) when stored in [restricted private transactions](#).
- **[P] PRIV-030:** [Enterprise Ethereum clients](#) *MUST* encrypt [payload data](#) when in transit in [restricted private transactions](#).
- **[P] PRIV-040:** [Enterprise Ethereum clients](#) *MAY* encrypt [metadata](#) when stored in [restricted private transactions](#).
- **[P] PRIV-050:** [Enterprise Ethereum clients](#) *MAY* encrypt [metadata](#) when in transit in [restricted private transactions](#).
- **[P] PRIV-060:** [Nodes](#) that relay a [restricted private transaction](#), but are not party to that [transaction](#), *MUST NOT* store the [payload data](#).
- **[P] PRIV-070:** [Nodes](#) that relay a [restricted private transaction](#), but are not party to that [transaction](#), *SHOULD NOT* store the [metadata](#).
- **[P] PRIV-080:** The implementation of the **eea\_sendTransactionAsync**, **eea\_sendTransaction**, **eea\_sendRawTransactionAsync**, or **eea\_sendRawTransaction** methods (see Section § 6.3.2 [Extensions to the JSON-RPC API](#)) with the **restriction** parameter set to **restricted**, *MUST* result in a [restricted private transaction](#).

## NOTE: Restricted Private Transactions

Private transactions can be implemented by creating private channels, that is, private smart contracts where the payload data is only stored by the clients participating in a transaction, and not by any other client (despite that the payload data might be encrypted and only decodable by authorized parties).

Private transactions are kept private between related parties, so unrelated parties have no access to the content of the transaction, the sending party, or the addresses of accounts party to the transaction. In fact, a private smart contract has a similar relationship to the blockchain that hosts it as a private blockchain that is only replicated and certified by a subset of participating nodes, but is notarized and synchronized on the hosting blockchain. This private blockchain is thus able to refer to data in less restrictive private smart contracts, as well as in public smart contracts.

If implementing unrestricted private transactions:

- [P] PRIV-090: Enterprise Ethereum clients *SHOULD* encrypt the recipient identity when stored in unrestricted private transactions.
- [P] PRIV-100: Enterprise Ethereum clients *SHOULD* encrypt the sender identity when stored in unrestricted private transactions.
- [P] PRIV-110: Enterprise Ethereum clients *SHOULD* encrypt the payload data when stored in unrestricted private transactions.
- [P] PRIV-120: Enterprise Ethereum clients *MUST* encrypt payload data when in transit in unrestricted private transactions.
- [P] PRIV-130: Enterprise Ethereum clients *MAY* encrypt metadata when stored in unrestricted private transactions.
- [P] PRIV-140: Enterprise Ethereum clients *MAY* encrypt metadata when in transit in unrestricted private transactions.
- [P] PRIV-150: Nodes that relay an unrestricted private transaction, but are not party to that transaction, *MAY* store the payload data.
- [P] PRIV-160: Nodes that relay an unrestricted private transaction, but are not party to that transaction, *MAY* store the metadata.
- [P] PRIV-170: The implementation of the `eea_sendTransactionAsync`, `eea_sendTransaction`, `eea_sendRawTransactionAsync`, or `eea_sendRawTransaction` methods (see Section § 6.3.2 Extensions to the JSON-RPC API) with the `restriction` parameter set to `unrestricted` *MUST* result in an unrestricted private transaction.



- **[P] PRIV-210:** Enterprise Ethereum clients implementation of unrestricted private transactions *MUST* provide the ability for nodes to achieve global consensus.

**NOTE: Unrestricted Private Transactions**

Obfuscated data that is replicated across all nodes can be reconstructed by any node, albeit in encrypted form. Mathematical transactions on numerical data are intended to be validated by the underlying Enterprise Ethereum blockchain on a zero-knowledge basis. The plaintext content is only available to participating parties to the transaction. Thus, a node is expected to have the ability to maintain and transact against numerical balances certified by the whole community of validators on a zero-knowledge basis.

An alternative to the zero-knowledge approach could be the combined use of ring signatures, stealth addresses, and mixing, which is demonstrated to provide the necessary level of obfuscation that is mathematically impossible to penetrate and does not rely on the trusted setup required by ZK-SNARKS.

**[P] PRIV-180:** Enterprise Ethereum clients *SHOULD* be able to extend the set of parties privy to a private transaction (or forward the private transaction in some way).

**[P] PRIV-190:** Enterprise Ethereum clients *SHOULD* provide the ability for nodes to achieve consensus on their mutually private transactions.

The differences between restricted private transactions and unrestricted private transactions are summarized in the table below.

*Table 2 Restricted and Unrestricted Private Transactions*

Restricted Private TXNs (if implemented)		Unrestricted Private TXNs (if implemented)	
Metadata	Payload Data	Metadata	Payload Data
<i>MAY</i> encrypt	<i>MUST</i> encrypt	<i>MAY</i> encrypt <i>SHOULD</i> encrypt sender and recipient identity	<i>MUST</i> encrypt in transit <i>SHOULD</i> encrypt in storage
<i>SHOULD NOT</i> allow storage by non-participating <u>nodes</u>	<i>MUST NOT</i> allow storage by non-participating <u>nodes</u>	<i>MAY</i> allow storage by non-participating <u>nodes</u>	<i>MAY</i> allow storage by non-participating <u>nodes</u>

## 7.1.4 Privacy Groups

*This section is non-normative.*

A **privacy group** is the collection of participants privy to a [private transaction](#). Each member of the group has the ability to decrypt and read a [private transaction](#) sent to the group.

An [Enterprise Ethereum client](#) maintains the public world state for the blockchain and a private state for each [privacy group](#). The private states contain data that is not shared in the globally replicated world state. A [private transaction](#) causes a state transition in the public state (that is, a commitment of a [private transaction](#) occurred) and a state transition in the private state (that is, a [smart contract](#) state was changed or some information was exchanged in the private state).

The **privateFrom** and **privateFor** parameters in the send transaction calls are the public keys for the participants intended to be able to decrypt the [private transaction](#). A [privacy group](#) is given a unique [privacy group](#) ID. Members of a [privacy group](#) are specified by their public keys.

A [client](#) is expected to propagate a newly created or updated [privacy group](#) to the other members which are part of the [privacy group](#).

### EXAMPLE 2: Privacy Group example object

```
{
  "name": "my privacy group"
  "privacyGroupId": "Vbj70zF+G2V/8XoyZzwqawfcQ+r9BkXoLQ0qkQideys="
  "members": [
    "negmDcN2P40Dpqn/6WkJ02zT/0w0bjhGpkZ8UP6vARk=",
    "g59BmTeJIn7HIcnq8VQWgyh/pDbvbt2eyP0Ii60aDDw="
  ]
  "description": "this is an example privacy group"
}
```

## 7.2 Performance Sublayer

*This section is non-normative.*

Performance is an important requirement for [Enterprise Ethereum clients](#) as many use cases for [Enterprise Ethereum blockchains](#) imply a high volume of [transactions](#), or computationally heavy tasks. A blockchain's overall performance is constrained by the slowest [node](#).

There are many different aspects of performance. Instead of mandating specific requirements, this Specification notes the importance of performance, leaving [Enterprise Ethereum client](#) developers free to implement whatever strategies are appropriate for their software.

This Specification does not constrain experimentation to improve the performance of [Enterprise Ethereum clients](#). This is an active area of research, and it is likely various techniques to improve performance will be developed over time, which cannot be exactly predicted.

This Specification does mandate or allow for several optimizations to improve performance. The most important techniques maximize the throughput of [transactions](#).

### 7.2.1 On-chain (Layer 1 and Layer 2) Scaling

Techniques to improve performance through scaling are valuable for blockchains with high [transaction](#) throughput requirements that keep the processing on the blockchain.

On-chain (layer 1) scaling techniques, like [\[sharding\]](#), are changes or extensions to the [public Ethereum](#) protocol to facilitate increased [transaction](#) speeds.

On-chain (layer 2) scaling techniques use [smart contracts](#), and approaches like [\[Plasma\]](#), or [\[state-channels\]](#), to increase [transaction](#) speed without changing the underlying [Ethereum](#) protocol. For more information, see [\[Layer2-Scaling-Solutions\]](#).

### 7.2.2 Off-chain (Layer 2 Compute)

[Off-chain Computing](#) can be used to increase [transaction](#) speeds, by moving the processing of computationally intensive tasks from [nodes](#) processing [transactions](#) to one or more [Trusted Computing services](#), reducing the resources needed by [nodes](#) and allowing them to produce blocks faster. This functionality can be implemented by [Enterprise Ethereum clients](#) implementing requirement [EXEC-050](#).

## 7.3 Permissioning Sublayer

**Permissioning** is the property of a system that ensures operations are executed by and accessible to designated parties. For [Enterprise Ethereum](#), permissioning refers to the ability of a [node](#) to join an [Enterprise Ethereum blockchain](#), and the ability of individual [accounts](#) or [nodes](#) to perform specific functions. For example, an [Enterprise Ethereum blockchain](#) might only allow certain [nodes](#) to act as validators, and only certain [accounts](#) to instantiate [smart contracts](#).

[Enterprise Ethereum](#) provides a [permissioned](#) implementation of [Ethereum](#) supporting peer [node](#) connectivity [permissioning](#), [account permissioning](#), and [transaction](#) type [permissioning](#).

### 7.3.1 Nodes

[C] **NODE-010:** [Enterprise Ethereum](#) implementations *MUST* provide the ability to specify at startup a list of static peer [nodes](#) to establish peer-to-peer connections with.

[C] **NODE-020:** [Enterprise Ethereum clients](#) *MUST* provide the ability to enable or disable peer-to-peer [node](#) discovery.

[P] **NODE-030:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify a whitelist of the [nodes](#) permitted to connect to a [node](#).

[P] **NODE-080:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify [node](#) identities in a way aligned with the concept of [groups](#).

[P] **NODE-090:** [Enterprise Ethereum clients](#) *MUST* document which metadata parameters (if any) can affect [transaction](#) ordering, and what the effects are.

### 7.3.2 Ethereum Accounts

For the purpose of this Specification:

- An **organization** is a logical [group](#) composed of Ethereum [accounts](#), [nodes](#), and other organizations or suborganization. A **suborganization** is an organization controlled by and subordinate to another organization. An organization typically represents an enterprise, or some identifiable part of an enterprise. For the purpose of [permissioning](#), organizations roughly correspond to the UNIX concept of [groups](#).
- A **user** is a human or an automated process interacting with an [Enterprise Ethereum blockchain](#) using the [Ethereum JSON-RPC API](#). The identity of a user is represented by an [Ethereum account](#). Public key cryptography is used to sign [transactions](#) made by the user so the [EVM](#) can authenticate the identity of a user sending a [transaction](#).
- An **Ethereum account** is an established relationship between a [user](#) and an [Ethereum blockchain](#). Having an Ethereum account allows [users](#) to interact with a blockchain, for example to submit [transactions](#) or deploy [smart contracts](#). See also [wallet](#).
- **Groups** are collections of [users](#) that have or are allocated one or more common attributes. For example, common privileges allowing [users](#) to access a specific set of services or functionality.
- **Roles** are sets of administrative tasks, each with associated [permissions](#) that apply to [users](#) or administrators of a system, used for example in RBAC [permissioning contracts](#).

[P] **PART-010:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify a whitelist of [accounts](#) that are permitted to transact with the blockchain.

[P] **PART-015:** [Enterprise Ethereum clients](#) *MUST* be able to verify that [accounts](#) are present on the whitelist required by [PART-010](#): when adding [transactions](#) from the [account](#) to a block, and

when verifying a received block containing [transactions](#) created by that [account](#).

**[P] PART-050:** [Enterprise Ethereum clients](#) *MUST* provide a mechanism to identify [organizations](#) that participate in the [Enterprise Ethereum blockchain](#).

#### NOTE

A specific mechanism to identify [organizations](#) could be identified in a future version of this Specification.

**[P] PART-055** [Enterprise Ethereum clients](#) *MUST* support anonymous [accounts](#).

**[P] PART-060:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify [accounts](#) in a way aligned with the concepts of [groups](#) and [roles](#).

**[P] PART-070:** [Enterprise Ethereum clients](#) *MUST* be able to authorize the types of [transactions](#) an [account](#) can submit, providing separate [permissioning](#) for the ability to:

- Deploy [smart contracts](#).
- Call functions that change the state of specified [smart contracts](#).
- Perform a value transfer to a specified [account](#).

#### NOTE

Because deep nesting of structures can introduce unacceptable performance issues, implementations can limit the number of nesting levels they enable. This Specification defines a minimum requirement, although in practice the number of levels implementations support is not constrained to any specific value, and depends entirely on implementation choices.

**[C] PERM-075:** [Enterprise Ethereum clients](#) *MUST* allow [organizations](#) to be nested to a minimum of three levels. That is, an [organization](#) that contains an [organization](#) that contains another [organization](#).

### 7.3.3 Additional Permissioning Requirements

**[C] PERM-020:** [Enterprise Ethereum clients](#) *SHOULD* provide the ability for [network configuration](#) to be updated at run time without the need to restart.

## 8. Core Blockchain Layer

The Consensus sublayer provides a mechanism to establish [consensus](#) between [nodes](#).

**Consensus** is the process of [nodes](#) on a blockchain reaching agreement about the current state of the blockchain.

A **consensus algorithm** is the mechanism by which a blockchain achieves [consensus](#). Different blockchains can use different consensus algorithms, but all [nodes](#) of a given blockchain need to use the same consensus algorithm. Different consensus algorithms are available for both [public Ethereum](#) and [Enterprise Ethereum](#).

[Enterprise Ethereum clients](#) can provide additional [consensus algorithms](#) for operations within their private **consortium network** (an [Ethereum](#) blockchain, either [public Ethereum](#) or [Enterprise Ethereum](#), which is not part of the [Ethereum MainNet](#)).

### EXAMPLE 3: Consensus Algorithms

An example public [consensus algorithm](#) is the Proof of Work (PoW) algorithm, which is described in the [\[Ethereum-Yellow-Paper\]](#). Over time, PoW is likely to be phased out from use and replaced with new methods of [consensus](#). Other example [consensus algorithms](#) include Istanbul [\[Byzantine-Fault-Tolerant\]](#) (IBFT) [\[EIP-650\]](#), [\[RAFT\]](#), and Proof of Elapsed Time [\[PoET\]](#).

The Execution sublayer implements the **Ethereum Virtual Machine** (EVM), which is a runtime computing environment for the execution of [smart contracts](#). Each [node](#) operates an EVM.

Ethereum-flavored WebAssembly [\[eWASM\]](#), which has its own instruction set, and other computational capabilities as required, are implemented at this layer.

**Smart contracts** are computer programs that the [EVM](#) executes. Smart contracts can be written in higher-level programming languages and compiled to [EVM](#) bytecode. Smart contracts can implement a contract between parties, where the execution is guaranteed and auditable to the level of security provided by [Ethereum](#) itself.

A **precompiled contract** is a [smart contract](#) compiled in [EVM](#) bytecode and stored by a [node](#).

Finally, the Storage and Ledger sublayer is provided to store the blockchain state, such as [smart contracts](#) for later execution. This sublayer follows blockchain security paradigms such as using cryptographically hashed tries, a UTXO model, or at-rest-encrypted key-value stores.

## 8.1 Storage and Ledger Sublayer

To operate a [client](#) on the [Ethereum MainNet](#), and to support optional off-chain operations, local data storage is required. For example, [Enterprise Ethereum clients](#) can locally cache the results

from a trusted [oracle](#) or store information related to systems extensions that are beyond the scope of this Specification.

[C] **STOR-030:** [Enterprise Ethereum clients](#) providing support for multiple blockchains (for example, more than one [Enterprise Ethereum blockchain](#), or a public network) *MUST* store data related to restricted [private transactions](#) for those blockchains in [private state](#) dedicated to the relevant blockchain.

*Private State* is the state data that is not shared in the clear in the globally replicated state tree. This data can represent bilateral or multilateral arrangements between parties, for example in [private transactions](#).

[P] **STOR-040:** [Enterprise Ethereum clients](#) *SHOULD* permit a [smart contract](#) operating on [private state](#) to access [private state](#) created by other [smart contracts](#) involving the same parties to the [transaction](#).

[P] **STOR-050:** [Enterprise Ethereum clients](#) *MUST NOT* permit a [smart contract](#) operating on [private state](#) to access [private state](#) created by other [smart contracts](#) involving different parties to the [transaction](#).

[P] **STOR-070:** If an [Enterprise Ethereum client](#) stores [private state](#) persistently, it *SHOULD* protect the data using an Authenticated Encryption with Additional Data (AEAD) algorithm, such as one described in [\[RFC5116\]](#).

## 8.2 Execution Sublayer

[P] **EXEC-010:** [Enterprise Ethereum clients](#) *MUST* provide a [smart contract](#) execution environment implementing the [public Ethereum EVM](#) opcode set [\[EVM-Opcodes\]](#).

[P] **EXEC-020:** [Enterprise Ethereum clients](#) that provide a [smart contract](#) execution environment extending the [public Ethereum EVM](#) opcode set [\[EVM-Opcodes\]](#) *MUST* register the opcode and the name of the [Enterprise Ethereum client](#) in the [\[EEA-extended-opcode-registry\]](#).

[P] **EXEC-025:** [Enterprise Ethereum clients](#) that provide a [smart contract](#) execution environment extending the [public Ethereum EVM](#) opcode set [\[EVM-Opcodes\]](#) *SHOULD* register a description of the new functionality, and a URL for a complete specification and test suites in the [\[EEA-extended-opcode-registry\]](#), and create an EIP describing the new opcode.

[P] **EXEC-030:** [Enterprise Ethereum clients](#) *SHOULD* support the ability to synchronize their public state with the public state held by other [public Ethereum nodes](#).

[P] **EXEC-040:** [Enterprise Ethereum clients](#) *SHOULD* support compilation, storage, and execution of [precompiled contracts](#).



Trusted Computing ensures only authorized parties can execute smart contracts on an execution environment available to a given Enterprise Ethereum blockchain.

[C] EXEC-050: Enterprise Ethereum clients *MAY* support off-chain Trusted Computing

Multiple encryption techniques can be used to secure Trusted Computing and private state.

[C] EXEC-060: Enterprise Ethereum clients *MAY* support configurable alternative cryptographic curves as encryption options for Enterprise Ethereum blockchains.

### 8.2.1 Finality

**Finality** occurs when a transaction is definitively part of the blockchain and cannot be removed. A transaction reaches finality after some event defined for the relevant blockchain occurs. For example, an elapsed amount of time or a specific number of blocks added.

[P] FINL-010: When a deterministic consensus algorithm is used, Enterprise Ethereum clients *SHOULD* treat transactions as final after a defined interval or event. For example, after a defined time period has elapsed, or after a defined number of blocks were created since the transaction was included in a block.

## 8.3 Consensus Sublayer

A common consensus algorithm implemented by all clients is required to ensure interoperability between clients.

[Byzantine-Fault-Tolerant] consensus algorithms ensure a certain proportion of malfunctioning nodes performing voting, block-making, or validation roles do not pose a critical risk to the blockchain. This makes them an excellent choice for many blockchains. The Technical Specification Working Group are considering existing and new Byzantine-Fault-Tolerant consensus algorithms, primarily those related to IBFT [EIP-650], with the goal of adopting the outcomes of that work as a required consensus algorithm as soon as possible.

[P] CONS-030: One or more consensus algorithms *SHOULD* allow operations as part of an Enterprise Ethereum blockchain.

[P] CONS-050: Enterprise Ethereum clients *MAY* implement multiple consensus algorithms and use them on sidechain networks.

A **sidechain** is a separate Ethereum blockchain operating on the Enterprise Ethereum blockchain nodes. A sidechain can be public or private and can also operate on a consortium network.



[P] **CONS-093:** [Enterprise Ethereum clients](#) *MUST* support the Clique, Proof of Authority consensus algorithm [\[EIP-225\]](#).

[P] **CONS-110:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify the [consensus algorithms](#), through [network configuration](#), to be used for each public blockchain, private blockchain, and [sidechain](#) in use.

## 9. Network Layer

The Network layer consists of an implementation of a peer-to-peer networking protocol allowing [nodes](#) to communicate with each other. For example, the *DEVp2p* protocol, which defines messaging between [nodes](#) to establish and maintain a communications channel for use by higher layer protocols.

### 9.1 Network Protocol Sublayer

Network protocols define how [nodes](#) communicate with each other.

[P] **PROT-010:** [Nodes](#) *MUST* be identified and advertised using the [Ethereum](#) [\[enode\]](#) URL format.

[P] **PROT-015:** [Enterprise Ethereum clients](#) *MUST* implement the [\[DEVp2p-Node-Discovery\]](#) protocol.

The [\[Ethereum-Wire-Protocol\]](#) defines higher layer protocols, known as *capability protocols*, for messaging between [nodes](#) to exchange status, including block and [transaction](#) information. [\[Ethereum-Wire-Protocol\]](#) messages are sent and received over an already established [DEVp2p](#) connection between [nodes](#).

[P] **PROT-020:** [Enterprise Ethereum clients](#) *MUST* use the [\[DEVp2p-Wire-Protocol\]](#) for messaging between [nodes](#) to establish and maintain a communications channel for use by [capability protocols](#).

[P] **PROT-040:** [Enterprise Ethereum clients](#) *MAY* add new protocols or extend existing [Ethereum](#) protocols.

[P] **PROT-050:** To minimize the number of point-to-point connections needed between private [nodes](#), some private [nodes](#) *SHOULD* be capable of relaying [private transaction](#) data to multiple other private [nodes](#).

#### EXAMPLE 4: Relaying Private Transaction Data

Multi-party private [smart contracts](#) and [transactions](#) do not require direct connectivity between all parties because this is very impractical in enterprise settings, especially when many parties are allowed to [transact](#). [Nodes](#) common to all parties (for example, voters or blockmakers acting as bootnodes to all parties, and as backup or disaster recovery [nodes](#)) are intended to function as gateways to synchronize private [smart contracts](#) transparently. [Transactions](#) on private [smart contracts](#) could then be transmitted to all participating parties in the same way.

[P] **PROT-060:** [Enterprise Ethereum clients](#) *SHOULD* implement the [[Whisper-protocol](#)].

[P] **PROT-070:** [Enterprise Ethereum clients](#) *MUST* interpret the [parameters defined in this specification](#sec-code-definitions) for [network configuration](#) when found in a genesis.json file.

*Network configuration* refers to the [collection of settings defined for a blockchain](#sec-code-definitions), such as which [consensus algorithm](#) to use, or the addresses of [permissioning contracts](#). It is a set of parameters included as JSON data in a `genesis.json` file.

## 10. Anti-spam

This section refers to mechanisms to prevent the [Enterprise Ethereum blockchain](#) being degraded with a flood of intentional or unintentional [transactions](#). This might be realized through interfacing with an external security manager, as described in Section § 6.2.1 [Enterprise Management Systems](#), or implemented by the [client](#), as described in the following requirement.

[P] **SPAM-010:** [Enterprise Ethereum clients](#) *SHOULD* provide effective anti-spam mechanisms so attacking [nodes](#) or [accounts](#) (either malicious, buggy, or uncontrolled) can be quickly identified and stopped.

#### EXAMPLE 5: Anti-spam Mechanisms

Anti-spam mechanisms might include:

- Stopping parties attempting to issue [transactions](#) above a threshold volume.
- Providing a mechanism to enforce a cost for [gas](#), so [transacting](#) parties have to acquire and pay for (or destruct) private ether to [transact](#).
- Having a dynamic cost of [gas](#) based on activity intensity.

## 11. Cross-client Compatibility

Cross-client compatibility refers to the ability of an [Enterprise Ethereum blockchain](#) to operate with different [clients](#).

This Specification extends the capabilities and interfaces of [public Ethereum](#). The requirements relating to supporting and extending the [public Ethereum](#) opcode set are outlined in Section [§ 8.2 Execution Sublayer](#).

**[P] XCLI-005:** Features of [public Ethereum](#) implemented in [Enterprise Ethereum clients](#) *MUST* be compatible with the Constantinople [hard fork](#) of [Ethereum](#) [EIP-1013], which occurred on 28 February, 2019.

Future versions of this Specification are expected to align with newer versions of [public Ethereum](#) as they are deployed.

**[P] XCLI-020:** [Enterprise Ethereum clients](#) *MAY* extend the [public Ethereum](#) APIs. To maintain compatibility, [Enterprise Ethereum clients](#) *SHOULD* ensure these new features are a superset of the [public Ethereum](#) APIs.

#### **EXAMPLE 6:** Extensions to the Public Ethereum API

Extensions to [public Ethereum](#) APIs could include peer-to-peer APIs, the [JSON-RPC-API] over IPC, HTTP/HTTPS, or websockets.

**[P] XCLI-030:** [Enterprise Ethereum clients](#) *MUST* implement the [gas](#) mechanism specified in the [Ethereum-Yellow-Paper].

**[P] XCLI-040:** [Enterprise Ethereum clients](#) *MUST* function correctly when the [Gas](#) price is set to zero.

**[P] XCLI-050:** [Enterprise Ethereum clients](#) *MUST* implement the eight precompiled contracts defined in Appendix E of the [Ethereum-Yellow-Paper]:

- [ecrecover](#)
- [sha256hash](#)
- [ripemd160hash](#)
- [dataCopy](#)
- [bigModExp](#)
- [bn256Add](#)
- [bn256ScalarMul](#)
- [bn256Pairing](#)

## NOTE

Sample [\[implementation-code-in-Golang\]](#), as part of the Go-Ethereum client is available from the Go-Ethereum source repository [\[geth-repo\]](#). Be aware this code uses a combination of GPL3 and LGPL3 licenses

Cross-client compatibility extends to the different message encoding formats used by [clients](#).

[P] **XCLI-055:** [Enterprise Ethereum clients](#) *MUST* register precompiled contracts following the mechanisms defined by [\[EIP-1352\]](#):

[P] **XCLI-060:** [Enterprise Ethereum clients](#) *MUST* support the Contract Application Binary Interface ([\[ABI\]](#)) for interacting with [smart contracts](#).

[P] **XCLI-070:** [Enterprise Ethereum clients](#) *MUST* support Recursive Length Prefix ([\[RLP\]](#)) encoding for binary data.

## 12. Cross-chain Interoperability

*This section is non-normative.*

**Cross-chain interoperability** broadly refers to the ability to consume data from another chain (read) and to cause an update or another transaction on a distinct chain (write).

Cross-chain interoperability can take two forms:

- [Ethereum](#) to [Ethereum](#) (for example, two or more logically distinct [EVM](#)-based chains)
- [Ethereum](#) to another blockchain architecture.

[Cross-chain interoperability](#) is seen as a valuable feature by both the [Enterprise Ethereum](#) community and outside. Users of blockchain and blockchain-inspired platforms want to make use of data and functionality on heterogenous platforms.

The goals for [cross-chain interoperability](#) in this specification are to:

- Describe the layers of interoperability that are relevant to [Enterprise Ethereum](#) blockchains.
- Enable data consumption between different blockchains without using a trusted intermediary.
- Allow transaction execution across blockchains without a trusted intermediary.

## 13. Synchronization and Disaster Recovery

*This section is non-normative.*

Synchronization and disaster recovery refers to how [nodes](#) in a blockchain behave when connecting for the first time or reconnecting.

Various techniques can help do this efficiently. For an [Enterprise Ethereum blockchain](#) with few copies, off-chain backup information can be important to ensure the long-term existence of the information stored. A common backup format helps increase [client interoperability](#).

## A. Additional Information

### A.1 Terms defined in this specification

[account](#) see Ethereum account

[account permissioning](#)

[capability protocols](#)

[Client requirements](#)

[Consensus](#)

[consensus algorithm](#)

[consortium network](#)

[cross-chain interoperability](#)

[DApps](#)

[DEVp2p](#)

[Enterprise Ethereum](#)

[Enterprise Ethereum blockchains](#)

[Enterprise Ethereum client](#)

[Ethereum account](#)

[Ethereum JSON-RPC API](#)

[Ethereum MainNet](#)

[Ethereum Name Service](#)

[Ethereum Virtual Machine](#)

[Finality](#)

[Formal verification](#)

[Gas](#)

[genesis block](#)

[Groups](#)

[hard fork block](#)

[hard fork](#)

[Integration libraries](#)

[inter-chain mediators](#)

[interoperability](#)

[MainNet](#)

[Metadata](#)

[network configuration](#)

[node](#)

[node permissioning](#)

[Off-chain trusted computing](#)

[oracle](#)

[organization](#)

[Payload data](#)

[permissioning contracts](#)

[permissioning enforcement](#)

[permissioning management](#)

[Permissioning](#)

[precompiled contract](#)

[Private State](#)

[private transaction](#)

[private transaction manager](#)

[Protocol requirements](#)

[Public Ethereum](#)

[Restricted private transactions](#)

[Roles](#)

[sidechain](#)

[Smart contract languages](#)

[Smart contracts](#)

[transaction](#)

[Unrestricted private transactions](#)

[User](#)

[Wallets](#)

[zero-knowledge proof](#)

## A.2 Events, functions, methods, parameters defined in this specification

- [addsPermissions](#) parameter of [permissionsUpdated](#) events
- [addsRestrictions](#) parameter of [permissionsUpdated](#) events
- [connectionAllowed](#) function
- [nodePermissionContract](#) [network configuration](#) parameter
- [AccountPermissionsUpdated](#) event
- [NodePermissionsUpdated](#) event
- [permissionsUpdated](#) events
- [transactionAllowed](#) function
- [transactionPermissionContract](#) [network configuration](#) parameter

## A.3 Summary of Requirements

This section summarizes all of the requirements in this Specification into one section.

**[P] SMRT-030:** [Enterprise Ethereum clients](#) *MUST* support [smart contracts](#) of at least 24,576 bytes in size.

**[P] SMRT-040:** [Enterprise Ethereum clients](#) *MUST* read and enforce a size limit for [smart contracts](#) from the current [network configuration](#) (for example, from the [genesis block](#)).

**[P] SMRT-050:** If no contract size limit is specified in a [genesis block](#), subsequent [hard fork block](#), or [network configuration](#), [Enterprise Ethereum clients](#) *MUST* enforce a size limit on [smart contracts](#) of 24,576 bytes.

[P] JRPC-010: Enterprise Ethereum clients *MUST* provide support for the following Ethereum JSON-RPC API methods:

- `net_version`
- `net_peerCount`
- `net_listening`
- `eth_protocolVersion`
- `eth_syncing`
- `eth_coinbase`
- `eth_hashrate`
- `eth_gasPrice`
- `eth_accounts`
- `eth_blockNumber`
- `eth_getBalance`
- `eth_getStorageAt`
- `eth_getTransactionCount`
- `eth_getBlockTransactionCountByHash`
- `eth_getBlockTransactionCountByNumber`
- `eth_getCode`
- `eth_sendRawTransaction`
- `eth_call`
- `eth_estimateGas`
- `eth_getBlockByHash`
- `eth_getBlockByNumber`
- `eth_getTransactionByHash`
- `eth_getTransactionByBlockHashAndIndex`
- `eth_getTransactionByBlockNumberAndIndex`
- `eth_getTransactionReceipt`
- `eth_getUncleByBlockHashAndIndex`
- `eth_getUncleByBlockNumberAndIndex`
- `eth_getLogs`.

**[P] JRPC-007:** [Enterprise Ethereum clients](#) *SHOULD* implement [\[JSON-RPC-API\]](#) methods to be backward compatible with the definitions given in version f4e6248 of the [Ethereum JSON-RPC API](#) reference [\[JSON-RPC-API-vf4e6248\]](#), unless breaking changes were made and widely implemented for the health of the ecosystem. For example, to fix a major security or privacy problem.

**[C] JRPC-015:** [Enterprise Ethereum clients](#) *MUST* provide the capability to accept and respond to JSON-RPC method calls over a websocket interface.

**[C] JRPC-040:** [Enterprise Ethereum clients](#) *MUST* provide an implementation of the `debug_traceTransaction` method [\[debug-traceTransaction\]](#) from the Go Ethereum Management API.

**[C] JRPC-050:** [Enterprise Ethereum clients](#) *MUST* implement the [\[JSON-RPC-PUB-SUB\]](#) API.

**[P] JRPC-070:** [Enterprise Ethereum clients](#) implementing additional nonstandard subscription types for the [\[JSON-RPC-PUB-SUB\]](#) API *MUST* prefix their subscription type names with a namespace prefix other than `eea_`.

**[P] JRPC-080:** The [\[JSON-RPC\]](#) method name prefix `eea_` *MUST* be reserved for future use for RPC methods specific to the EEA.

**[P] JRPC-020:** [Enterprise Ethereum clients](#) *MUST* provide one of the following sets of extensions to create [private transaction](#) types defined in Section [§ 7.1.3 Private Transactions](#):

- `eea_sendTransactionAsync` and `eea_sendTransaction`, or
- `eea_sendRawTransactionAsync` and `eea_sendRawTransaction`.

**[P] JRPC-030:** The `eea_sendTransactionAsync`, `eea_sendTransaction`, `eea_sendRawTransactionAsync`, and `eea_sendRawTransaction` methods *MUST* respond with a [\[JSON-RPC\]](#) error response when an unimplemented [private transaction](#) type is requested. The error response *MUST* have the `code` `-50100` and the `message` `Unimplemented private transaction type`.

**[P] PERM-200:** [Enterprise Ethereum clients](#) *MUST* call the `connectionAllowed` function, as specified in Section [§ 6.3.3.3.1 Node Permissioning Functions](#), or if it implements [PERM-220](#) and [PERM-230](#), *MAY* use cached information to determine whether a connection with another [node](#) is permitted, and any restrictions to be placed on that connection.

**[P] PERM-210:** When checking the response to `connectionAllowed`, if any unknown permissioning bits are found to be zero, [Enterprise Ethereum clients](#) *MUST* reject the connection.

**[P] PERM-220:** On receipt of a `NodePermissionsUpdated` event containing an `addsRestrictions` property with the value `true`, [Enterprise Ethereum clients](#) *MUST* close any



network connections that are no longer permitted, and impose newly added restrictions on any network connections that have had restrictions added.

**[P] PERM-230:** On receipt of a [NodePermissionsUpdated](#) event containing an [addsPermissions](#) property with the value `false`, [Enterprise Ethereum clients](#) *MUST* check whether existing network connections have had their restrictions lifted and allow future actions that are now permitted.

**[P] PERM-240:** When validating or mining a block, [Enterprise Ethereum clients](#) *MUST* call the [transactionAllowed](#) function, as specified in Section § 6.3.3.4.1 [Account Permissioning Function](#), with worldstate as at the block's parent, or if it implements **[PERM-250](#req-perm-250)]** and **[PERM-260](#req-perm-260)]** *MAY* use cached information, to determine if a [transaction](#) is permitted in a block.

**[P] PERM-250:** On receipt of an [AccountPermissionsUpdated](#) event containing an [addsRestrictions](#) property with the value `true`, [Enterprise Ethereum clients](#) *MUST* purge all cached results from previous calls to [transactionAllowed](#) where the result returned was `true`.

**[P] PERM-260:** On receipt of an [AccountPermissionsUpdated](#) event containing an [addsPermissions](#) property with the value `true`, [Enterprise Ethereum clients](#) *MUST* purge all cached results from previous calls to [transactionAllowed](#) where the result returned was `false`.

**[C] ORCL-010:** [Enterprise Ethereum clients](#) *SHOULD* provide the ability to securely interact with [oracles](#) to send and receive external off-chain information.

**[P] PRIV-010:** [Enterprise Ethereum clients](#) *MUST* support [private transactions](#) using either [restricted private transactions](#) or [unrestricted private transactions](#).

When implementing [restricted private transactions](#):

- **[P] PRIV-020:** [Enterprise Ethereum clients](#) *MUST* encrypt [payload data](#) when stored in [restricted private transactions](#).
- **[P] PRIV-030:** [Enterprise Ethereum clients](#) *MUST* encrypt [payload data](#) when in transit in [restricted private transactions](#).
- **[P] PRIV-040:** [Enterprise Ethereum clients](#) *MAY* encrypt [metadata](#) when stored in [restricted private transactions](#).
- **[P] PRIV-050:** [Enterprise Ethereum clients](#) *MAY* encrypt [metadata](#) when in transit in [restricted private transactions](#).
- **[P] PRIV-060:** [Nodes](#) that relay a [restricted private transaction](#) but are not party to that transaction, *MUST NOT* store [payload data](#).
- **[P] PRIV-070:** [Nodes](#) that relay a [restricted private transaction](#) but are not party to that transaction *SHOULD NOT* store the [metadata](#).

- **[P] PRIV-080:** The implementation of the `eea_sendTransactionAsync`, `eea_sendTransaction`, `eea_sendRawTransactionAsync`, or `eea_sendRawTransaction` methods (see Section § 6.3.2 [Extensions to the JSON-RPC API](#)) with the `restriction` parameter set to `restricted`, *MUST* result in a [restricted private transaction](#).

When implementing [unrestricted private transactions](#):

- **[P] PRIV-090:** [Enterprise Ethereum clients](#) *SHOULD* encrypt the recipient identity when stored in [unrestricted private transactions](#).
- **[P] PRIV-100:** [Enterprise Ethereum clients](#) *SHOULD* encrypt the sender identity when stored in [unrestricted private transactions](#).
- **[P] PRIV-110:** [Enterprise Ethereum clients](#) *SHOULD* encrypt the [payload data](#) when stored in [unrestricted private transactions](#).
- **[P] PRIV-120:** [Enterprise Ethereum clients](#) *MUST* encrypt [payload data](#) when in transit in [unrestricted private transactions](#).
- **[P] PRIV-130:** [Enterprise Ethereum clients](#) *MAY* encrypt [metadata](#) when stored in [unrestricted private transactions](#).
- **[P] PRIV-140:** [Enterprise Ethereum clients](#) *MAY* encrypt [metadata](#) when in transit in [unrestricted private transactions](#).
- **[P] PRIV-150:** [Nodes](#) that relay an [unrestricted private transaction](#) but are not party to that transaction *MAY* store [payload data](#).
- **[P] PRIV-160:** [Nodes](#) that relay an [unrestricted private transaction](#) but are not party to that transaction, *MAY* store the [metadata](#).
- **[P] PRIV-170:** The implementation of the `eea_sendTransactionAsync`, `eea_sendTransaction`, `eea_sendRawTransactionAsync`, or `eea_sendRawTransaction` methods (see Section § 6.3.2 [Extensions to the JSON-RPC API](#)) with the `restriction` parameter set to `unrestricted` *MUST* result in an [unrestricted private transaction](#).
- **[P] PRIV-210:** [Enterprise Ethereum clients'](#) implementations of [unrestricted private transactions](#) *MUST* provide the ability for [nodes](#) to achieve global [consensus](#).

**[P] PRIV-180:** [Enterprise Ethereum clients](#) *SHOULD* be able to extend the set of parties to a [private transaction](#) (or forward the [private transaction](#) in some way).

**[P] PRIV-190:** [Enterprise Ethereum clients](#) *SHOULD* provide the ability for [nodes](#) to achieve [consensus](#) on their mutually [private transactions](#).

**[C] NODE-010:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify at startup a list of static peer [nodes](#) to establish peer-to-peer connections with.

**[C] NODE-020:** [Enterprise Ethereum clients](#) *MUST* provide the ability to enable or disable peer-to-peer [node](#) discovery.

**[P] NODE-030:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify a whitelist of the [nodes](#) permitted to connect to a [node](#).

**[P] NODE-080:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify [node](#) identities in a way aligned with the concept of [groups](#).

**[P] NODE-090:** [Enterprise Ethereum clients](#) *MUST* document which metadata parameters (if any) can affect [transaction](#) ordering, and what the effects are.

**[P] PART-010:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify a whitelist of [accounts](#) that are permitted to transact with the blockchain.

**[P] PART-015:** [Enterprise Ethereum clients](#) *MUST* be able to verify that [accounts](#) are present on the whitelist required by **PART-010**: when adding [transactions](#) from the [account](#) to a block, and when verifying a received block containing [transactions](#) created by that [account](#).

**[P] PART-050:** [Enterprise Ethereum clients](#) *MUST* provide a mechanism to identify [organizations](#) that participate in the [Enterprise Ethereum blockchain](#).

**[P] PART-055** [Enterprise Ethereum clients](#) *MUST* support anonymous [accounts](#).

**[P] PART-060:** [Enterprise Ethereum clients](#) *MUST* provide the ability to specify [accounts](#) in a way aligned with the concepts of [groups](#) and [roles](#).

**[P] PART-070:** [Enterprise Ethereum clients](#) *MUST* be able to authorize the types of [transactions](#) an [account](#) can submit, providing separate [permissioning](#) for the ability to:

- Deploy [smart contracts](#).
- Call functions that change the state of specified [smart contracts](#).
- Perform a value transfer to a specified [account](#).

**[C] PERM-075:** [Enterprise Ethereum clients](#) *MUST* allow [organizations](#) to be nested to a minimum of three levels. That is, an [organization](#) that contains an [organization](#) that contains another [organization](#).

**[C] PERM-020:** [Enterprise Ethereum clients](#) *SHOULD* provide the ability for [network configuration](#) to be updated at run time without the need to restart.

**[C] STOR-030:** [Enterprise Ethereum clients](#) providing support for multiple blockchains (for example, more than one [Enterprise Ethereum blockchain](#), or a public network) *MUST* store data related to restricted [private transactions](#) for those blockchains in [private state](#) dedicated to the relevant blockchain.

**[P] STOR-040:** Enterprise Ethereum clients *SHOULD* permit a smart contract operating on private state to access private state created by other smart contracts involving the same parties to the transaction.

**[P] STOR-050:** Enterprise Ethereum clients *MUST NOT* permit smart contract operating on private state to access private state created by other smart contracts involving different parties to the transaction.

**[P] STOR-070:** If an Enterprise Ethereum client stores private state persistently, it *SHOULD* protect the data using an Authenticated Encryption with Additional Data (AEAD) algorithm, such as one described in [RFC5116].

**[P] EXEC-010:** Enterprise Ethereum clients *MUST* provide a smart contract execution environment implementing the public Ethereum EVM opcode set [EVM-Opcodes].

**[P] EXEC-020:** Enterprise Ethereum clients that provide a smart contract execution environment extending the public Ethereum EVM opcode set [EVM-Opcodes] *MUST* register the opcode and name of the Enterprise Ethereum client in the [EEA-extended-opcode-registry].

**[P] EXEC-025:** Enterprise Ethereum clients that provide a smart contract execution environment extending the public Ethereum EVM opcode set [EVM-Opcodes] *SHOULD* register a description of the new functionality, and a URL for a complete specification and test suites in the [EEA-extended-opcode-registry], and create an EIP describing the new opcode.

**[P] EXEC-030:** Enterprise Ethereum clients *SHOULD* support the ability to synchronize their public state with the public state held by other public Ethereum nodes.

**[P] EXEC-040:** Enterprise Ethereum clients *SHOULD* support compilation, storage, and execution of precompiled contracts.

**[C] EXEC-050:** Enterprise Ethereum clients *MAY* support off-chain Trusted Computing

**[C] EXEC-060:** Enterprise Ethereum clients *MAY* support configurable alternative cryptographic curves as encryption options for Enterprise Ethereum blockchains.

**[P] FINL-010:** When a deterministic consensus algorithm is used, Enterprise Ethereum clients *SHOULD* treat transactions as final after a defined interval or event. For example, after a defined time period has elapsed, or after a defined number of blocks were created since the transaction was included in a block.

**[P] CONS-030:** One or more consensus algorithms *SHOULD* allow operations as part of an Enterprise Ethereum blockchain.

**[P] CONS-050:** Enterprise Ethereum clients *MAY* implement multiple consensus algorithms and use them on sidechain networks.

**[P] CONS-093:** Enterprise Ethereum clients *MUST* support the Clique, Proof of Authority consensus algorithm [EIP-225].

**[P] CONS-110:** Enterprise Ethereum clients *MUST* provide the ability to specify the consensus algorithms, through network configuration, to be used for each public blockchain, private blockchain, and sidechain in use.

**[P] PROT-010:** Nodes *MUST* be identified and advertised using the Ethereum enode URL format [enode].

**[P] PROT-015:** Enterprise Ethereum clients *MUST* implement the DEVp2p Node Discovery protocol [DEVp2p-Node-Discovery].

**[P] PROT-020:** Enterprise Ethereum clients *MUST* use the DEVp2p Wire Protocol [DEVp2p-Wire-Protocol] for messaging between nodes to establish and maintain a communications channel for use by capability protocols.

**[P] PROT-040:** Enterprise Ethereum clients *MAY* add new protocols or extend existing Ethereum protocols.

**[P] PROT-050:** To minimize the number of point-to-point connections needed between private nodes, some private nodes *SHOULD* be capable of relaying private transaction data to multiple other private nodes.

**[P] PROT-060:** Enterprise Ethereum clients *SHOULD* implement the [Whisper-protocol].

**[P] PROT-070:** Enterprise Ethereum clients *MUST* interpret the [parameters defined in this specification](#sec-code-definitions) for network configuration when found in a genesis.json file.

**[P] SPAM-010:** Enterprise Ethereum clients *SHOULD* provide effective anti-spam mechanisms so attacking nodes or accounts (either malicious, buggy, or uncontrolled) can be quickly identified and stopped.

**[P] XCLI-005:** Features of public Ethereum implemented in Enterprise Ethereum clients *MUST* be compatible with the Constantinople hard fork of Ethereum [EIP-1013], which occurred on 28 February, 2019.

**[P] XCLI-020:** Enterprise Ethereum clients *MAY* extend the public Ethereum APIs. To maintain compatibility, Enterprise Ethereum clients *SHOULD* ensure these new features are a superset of the public Ethereum APIs.

**[P] XCLI-030:** Enterprise Ethereum clients *MUST* implement the Gas mechanism specified in the [Ethereum-Yellow-Paper].

**[P] XCLI-040:** Enterprise Ethereum clients *MUST* function correctly when the Gas price is set to zero.

**[P] XCLI-050:** Enterprise Ethereum clients *MUST* implement the eight precompiled contracts defined in Appendix E of the [Ethereum-Yellow-Paper]:

- `ecrecover`
- `sha256hash`
- `ripemd160hash`
- `dataCopy`
- `bigModExp`
- `bn256Add`
- `bn256ScalarMul`
- `bn256Pairing`

**[P] XCLI-055:** Enterprise Ethereum clients *MUST* register precompiled contracts following the mechanisms defined by [EIP-1352]:

**[P] XCLI-060:** Enterprise Ethereum clients *MUST* support the Contract Application Binary Interface ([ABI]) for interacting with smart contracts.

**[P] XCLI-070:** Enterprise Ethereum clients *MUST* support Recursive Length Prefix ([RLP]) encoding for binary data.

## A.4 Acknowledgments

The EEA acknowledges and thanks the many people who contributed to the development of this version of the specification. Please advise us of any errors or omissions.

This version builds on the work of all who contributed to previous versions of the Enterprise Ethereum Client Specification, whom we hope are all acknowledged in those documents. We apologize to anyone whose name was left off the list. Please advise us at <https://entethalliance.org/contact/> of any errors or omissions.

We would also like to thank former editors David Hyland-Wood (version 1) and Daniel Burnett (version 2), and former EEA Technical Director, the late and missed Clifton Barber, for their work on previous versions of this specification.

Enterprise Ethereum is built on top of Ethereum, and we are grateful to the entire community who develops Ethereum, for their work and their ongoing collaboration to helps us maintain as much compatibility as possible with the Ethereum ecosystem.

## A.5 Changes



This section outlines substantive changes made to the specification since version 3:

- Update [PERM-200](#) and [PERM-240](#) to allow for caching information.
- Update the status of [eea\\_sendTransaction](#) and [eea\\_sendRawTransaction](#) from experimental to stable.
- Add [PERM-250](#) and [PERM-260](#) so [AccountPermissionsUpdated](#) events trigger appropriate cache invalidation.
- Add [PROT-070](#) requiring clients to interpret [network configuration](#) parameters as defined in this specification.
- Add [addsPermissions](#) parameter to the [permissionsUpdated](#) events in permissioning contracts. Modified [PERM-230](#) to use the new parameter.
- Move Permissioning Sublayer Section under new Enterprise 3P's Layer Section.
- Add [privacyGroupId](#) parameter to all [sendTransaction](#) JSON RPC calls. Add Privacy Groups sub section to Privacy and Scaling Layer Section.
- Removes Permissioning Smart Contract examples section. This has been copied to the separate Implementation Guide document.
- Update the definitions of [connectionAllowed](#) and [transactionAllowed](#) to note that they can be found at the address given by the relevant parameters of the [network configuration](#).
- Change [EXEC-060](#) to *MAY*, and clarify that it is about alternative crypto curves
- Reword [XCLI-040](#) as a requirement on the client to function when [Gas](#) price is zero.
- Remove permissioning requirements **NODE-040**, **NODE-050**, **NODE-060**, **PART-020**, **PART-025**, **PART-030** and **PART-040** as they are now requirements to be met by the [permissioning contract](#) on an [Enterprise Ethereum Blockchain](#).
- Remove Privacy Levels Section and Privacy Level Certification Section.
- Remove **DAPP-010** as it was not a client requirement.
- Change [res](#) field to [result](#) in node and account permissioning interface functions.
- Various updates to the architecture stack,.
- Move former requirements **PERM-040** and **PERM-050** to the [implementation guide](#).
- Update [EXEC-020](#) to require registration of extended opcodes, and add [\[P\] EXEC-025](#) suggesting registration of new opcode functionality and specification in the [\[EEA-extended-opcode-registry\]](#) and creating an EIP to describe the new opcode.
- Update definitions of [eea\\_sendTransactionAsync](#) and [eea\\_sendTransaction](#). Value must now be 0 if present. Also remove value from the examples. Update [privateFrom](#) and [privateFor](#) fields in examples, including changing [privateFor](#) to be an array.
- Remove the limit on the size of the DATA parameter in [eea\\_sendTransactionAsync](#) and [eea\\_sendTransaction](#).

- Change the length of the `privateFrom` parameter in `eea_sendTransaction*` from 20 to 32 bytes.
- Add **[P] XCLI-055:** requiring precompiled contracts to be registered according to [\[EIP-1352\]](#).
- Remove experimental `eea_clientCapabilities` RPC method.
- Update privacy requirements to require they "encrypt" rather than "support encryption of" private transaction data.

Note that similar sections in Version 2 and [version 3](#) describe the changes made to each version.

## B. References

### B.1 Normative references

#### [ABI]

*Contract ABI Specification*. Ethereum Foundation. URL: <https://solidity.readthedocs.io/en/develop/abi-spec.html>

#### [Byzantine-Fault-Tolerant]

*Byzantine Fault Tolerant*. URL: [https://en.wikipedia.org/wiki/Byzantine\\_fault\\_tolerance](https://en.wikipedia.org/wiki/Byzantine_fault_tolerance)

#### [debug-traceTransaction]

*debug\_traceTransaction*. URL: <https://github.com/ethereum/go-ethereum/wiki/Management-APIs>

#### [DEVp2p-Node-Discovery]

*Node Discovery Protocol*. URL: <https://github.com/ethereum/devp2p/blob/master/rlpx.md>

#### [DEVp2p-Wire-Protocol]

*DEVp2p Wire Protocol*. URL: <https://github.com/ethereum/devp2p/blob/master/caps/eth.md>

#### [EEA-extended-opcode-registry]

*EEA EVM opcode extensions registry*. Enterprise Ethereum Alliance, Inc. URL: <http://entethalliance.github.io/client-spec/extended-opcodes-registry.html>

#### [EIP-1013]

*Hardfork Meta: Constantinople*. Ethereum Foundation. URL: <https://eips.ethereum.org/EIPS/eip-1013>

#### [EIP-1352]

*Specify restricted address range for precompiles/system contracts*. Ethereum Foundation. URL: <https://eips.ethereum.org/EIPS/eip-1352>

#### [EIP-225]

*Clique proof-of-authority consensus protocol*. Ethereum Foundation. URL: <https://eips.ethereum.org/EIPS/eip-225>

#### [EIP-648]



*Easy Parallelizability*. Ethereum Foundation. URL:  
<https://github.com/ethereum/EIPs/issues/648>

**[EIP-650]**

*Istanbul Byzantine Fault Tolerance*. Ethereum Foundation. URL:  
<https://github.com/ethereum/EIPs/issues/650>

**[enode]**

*Ethereum enode URL format*. Ethereum Foundation. URL:  
<https://github.com/ethereum/wiki/wiki/enode-url-format>

**[Ethereum-Wire-Protocol]**

*Ethereum Wire Protocol*. URL: <https://github.com/ethereum/wiki/wiki/Ethereum-Wire-Protocol>

**[Ethereum-Yellow-Paper]**

*Ethereum: A Secure Decentralized Generalized Transaction Ledger*. Dr. Gavin Wood. URL:  
<https://ethereum.github.io/yellowpaper/paper.pdf>

**[EVM-Opcodes]**

*Ethereum Virtual Machine (EVM) Opcodes and Instruction Reference*. URL:  
<https://github.com/trailofbits/evm-opcodes>

**[eWASM]**

*Ethereum-flavored WebAssembly*. URL: <https://github.com/ewasm/design>

**[GDPR]**

*European Union General Data Protection Regulation*. European Union. URL: <https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1528874672298&uri=CELEX%3A32016R0679>

**[JSON]**

*The application/json Media Type for JavaScript Object Notation (JSON)*. D. Crockford. IETF. July 2006. Informational. URL: <https://tools.ietf.org/html/rfc4627>

**[JSON-RPC]**

*JavaScript Object Notation - Remote Procedure Call*. JSON-RPC Working Group. URL:  
<http://www.jsonrpc.org/specification>

**[JSON-RPC-API]**

*Ethereum JSON-RPC API*. Ethereum Foundation. URL:  
<https://github.com/ethereum/wiki/wiki/JSON-RPC>

**[JSON-RPC-API-vf4e6248]**

*Ethereum JSON-RPC API*. Ethereum Foundation. URL:  
<https://github.com/ethereum/wiki/wiki/JSON-RPC/f4e624855ae05371b3fb78e02f5052679063b0b2>

**[JSON-RPC-PUB-SUB]**

*RPC PUB-SUB*. Ethereum Foundation. URL: <https://github.com/ethereum/go-ethereum/wiki/RPC-PUB-SUB>

**[LLL]**

*LLL Introduction*. Ben Edgington. 2017. URL: [http://lll-docs.readthedocs.io/en/latest/lll\\_introduction.html](http://lll-docs.readthedocs.io/en/latest/lll_introduction.html)

**[Nethereum]**

*Nethereum .NET Integration Library*. Nethereum Open Source Community. URL: <https://nethereum.com>

**[Plasma]**

*Plasma: Scalable Autonomous Smart Contracts*. Joseph Poon and Vitalik Buterin. August 2017. URL: <https://plasma.io/plasma.pdf>

**[PSD2]**

*European Union Personal Service Directive*. European Union. URL: [https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366\\_en](https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366_en)

**[RFC2119]**

*Key words for use in RFCs to Indicate Requirement Levels*. S. Bradner. IETF. March 1997. Best Current Practice. URL: <https://tools.ietf.org/html/rfc2119>

**[RFC5116]**

*An Interface and Algorithms for Authenticated Encryption*. D. McGrew. IETF. January 2008. Proposed Standard. URL: <https://tools.ietf.org/html/rfc5116>

**[RFC8174]**

*Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words*. B. Leiba. IETF. May 2017. Best Current Practice. URL: <https://tools.ietf.org/html/rfc8174>

**[RLP]**

*Recursive Length Prefix*. Ethereum Foundation. URL: <https://github.com/ethereum/wiki/wiki/RLP>

**[sharding]**

*Sharding FAQs*. Ethereum Foundation. URL: <https://github.com/ethereum/wiki/wiki/Sharding-FAQs>

**[Solidity]**

*The Solidity Contract-Oriented Programming Language*. Ethereum Foundation. URL: <https://github.com/ethereum/solidity>

**[state-channels]**

*Counterfactual: Generalized State Channels*. URL: <https://counterfactual.com/statechannels>

**[web3.js]**

*Ethereum JavaScript API*. Ethereum Foundation. URL: <https://github.com/ethereum/web3.js>

**[web3j]**

*web3j Lightweight Ethereum Java and Android Integration Library*. Conor Svensson. URL: <https://web3j.io>

**[Whisper-protocol]**

*Whisper*. Ethereum Foundation. URL: <https://github.com/ethereum/wiki/wiki/Whisper>

**[ZK-STARKS]**

Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive. 2018-03-16. URL: <https://eprint.iacr.org/2018/046.pdf>

**[ZKP]**

Zero Knowledge Proof. Wikipedia. URL: [https://en.wikipedia.org/wiki/Zero-knowledge\\_proof](https://en.wikipedia.org/wiki/Zero-knowledge_proof)

## B.2 Informative references

**[EEA-implementation-guide]**

Enterprise Ethereum Alliance Implementation Guide V4.0. Enterprise Ethereum Alliance, Inc. URL: [https://entethalliance.org/wp-content/uploads/2018/11/EEA\\_Enterprise\\_Ethereum\\_Implementation\\_Guide\\_V4.pdf](https://entethalliance.org/wp-content/uploads/2018/11/EEA_Enterprise_Ethereum_Implementation_Guide_V4.pdf)

**[EEA-OC]**

EEA Off-Chain Trusted Compute Specification - Editors' draft. Enterprise Ethereum Alliance, Inc. URL: <https://entethalliance.github.io/trusted-computing/spec.html>

**[EIPs]**

Ethereum Improvement Proposals. Ethereum Foundation. URL: <https://eips.ethereum.org/>

**[ERC-20]**

Ethereum Improvement Proposal 20 - Standard Interface for Tokens. Ethereum Foundation. URL: <https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md>

**[ERC-223]**

Ethereum Improvement Proposal 223 - Token Standard. Ethereum Foundation. URL: <https://github.com/ethereum/EIPs/issues/223>

**[ERC-621]**

Ethereum Improvement Proposal 621 - Token Standard Extension for Increasing & Decreasing Supply. Ethereum Foundation. URL: <https://github.com/ethereum/EIPs/pull/621>

**[ERC-721]**

Ethereum Improvement Proposal 721 - Non-fungible Token Standard. Ethereum Foundation. URL: <https://github.com/ethereum/eips/issues/721>

**[ERC-827]**

Ethereum Improvement Proposal 827 - Extension to ERC-20. Ethereum Foundation. URL: <https://github.com/ethereum/EIPs/issues/827>

**[geth-repo]**

Go-Ethereum. URL: <https://github.com/ethereum/go-ethereum/>

**[implementation-code-in-Golang]**

implementation code in Golang. URL: <https://github.com/ethereum/go-ethereum/blob/master/core/vm/contracts.go#L50-L360>

**[Layer2-Scaling-Solutions]**

Making Sense of Ethereum's Layer 2 Scaling Solutions: State Channels, Plasma, and Truebit. Josh Stark. February 2018. URL: <https://medium.com/l4-media/making-sense-of-ethereums-layer-2-scaling-solutions-state-channels-plasma-and-truebit-22cb40dcc2f4>

**[PoET]**

*Proof of Elapsed Time 1.0 Specification*. Intel Corporation. 2015-2017. URL:  
<https://sawtooth.hyperledger.org/docs/core/releases/1.0/architecture/poet.html#>

**[RAFT]**

*Raft-based Consensus for Ethereum/Quorum*. J.P. Morgan. URL:  
<https://github.com/jpmorganchase/quorum/blob/master/raft/doc.md>

**[USECASES]**

*Use cases for Enterprise Ethereum Clients (EDITORS' DRAFT WORK IN PROGRESS)*. EEA Inc. URL: <https://entethalliance.github.io/client-spec/usecases.html>

**[WP-ABAC]**

*Attribute-based access control*. Wikipedia. URL: [https://en.wikipedia.org/wiki/Attribute-based\\_access\\_control](https://en.wikipedia.org/wiki/Attribute-based_access_control)

**[WP-RBAC]**

*Role-based access control*. URL: [https://en.wikipedia.org/wiki/Role-based\\_access\\_control](https://en.wikipedia.org/wiki/Role-based_access_control)