Workgroup: Network Working Group
Internet-Draft: draft-kowalik-rpp-architecture-00
:
Published: 7 March 2025
Intended Status: Informational
Expires: 8 September 2025
Authors: P. Kowalik M. Wullink
DENIC eG *SIDN Labs*

RPP Architecture

Abstract

Advancements in development, integration, deployment environments and operational paradigms have led to a desire for an alternatve for the Extensible Provisioning Protocol (EPP). This document defines the architecture for the RESTful Provisioning Protocol (RPP) an HTTP based provisioning protocol leveraging the REST architectural style and JSON data-interchange format, aiming to standardize a RESTful protocol for provisioning database objects. The architecture includes support for extensibility, allowing for multiple possible use cases. RPP is intended to co-exist with EPP, offering an alternative protocol including data model compatibility with EPP core objects and the benefits associated with the REST architectural style and widely adopted HTTP-based technologies.

Contributing

When contributing to this document, please use the following GitHub project: https://github.com/pawel-kow/RPP-architecture.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document outlines the architecture of the RESTful Provisioning Protocol (RPP). RPP aims to provide a modern, standardized, and developer-friendly protocol for provisioning and managing objects in a shared database or registry, initially focusing on functional equivalents of EPP object mappings for domain names [RFC5731], hosts [RFC5732], and contacts [RFC5733]. RPP also considers DNS provisioning as a potential use case, aiming for a uniform API layer for various registry operations.

RPP is designed to leverage the benefits of REST, including statelessness, ease of integration, and compatibility with existing web infrastructure and tooling such as OpenAPI, API gateways, and web application firewalls. By adopting JSON as the data-interchange format, RPP seeks to align with current development practices and the successful deployment patterns observed in protocols such as RDAP [RFC9082]. The choice for REST and JSON also facilitates direct browser and mobile application integration including modern security mechanisms such as OAuth2.0.

This architecture document serves as a foundation for a series of specifications that will collectively define RPP. It details the layered approach, core components, and design considerations for building an interoperable and extensible provisioning protocol. RPP is intended to coexist with EPP, offering an alternative for implementers seeking a RESTful approach without aiming to replace EPP or define migration paths from EPP. RPP aims for data model compatibility with EPP core objects to allow automatic and mechanical mapping and conversion, especially for core objects (domain, contact, host).

2. Terminology

This document uses terminology from RFC5730 [RFC5730] and broadly adopts the REST architectural principles as defined in [REST] and related RFCs.

- **RPP:** RESTful Provisioning Protocol. The protocol being defined by the RPP working group.
- **EPP:** Extensible Provisioning Protocol as defined in [RFC5730].
- **REST:** REpresentational State Transfer architectural style [REST].
- **JSON:** JavaScript Object Notation [RFC8259].
- **JWT:** JSON Web Token [RFC7519].
- **JWT-SD:** JWT with Selective Disclosure [I-D.draft-ietf-oauth-selective-disclosure-jwt].
- **OpenAPI:** The OpenAPI Specification (OAS) (formerly known as Swagger Specification) is an API description format for REST APIs.

3. Requirements

This document is based on the requirements defined by RPP WG in state from 7.3.2025 [RPPReq].

The actual state of the requirements is present on WG Wiki https://wiki.ietf.org/en/group/rpp/requirements.

4. Architectural Overview

This chapter provides an overview of the RESTful Provisioning Protocol (RPP) architecture. A key design principle is to leverage existing web standards and principles, particularly HTTP and REST principles. This allows RPP to delegate functionality and features to the well-established infrastructure and semantics of the web, focusing its own definitions on the specific domain of object provisioning. Therefore, we assume:

- **HTTP and RESTful principles are foundational:** RPP leverages HTTP for transport and adheres to RESTful principles for resource management.
- **Domain-specific logic resides in data representations:** The specifics of resource provisioning are encoded within the data structures and semantics of the RPP message bodies.
- **Layered architecture for modularity:** The architecture is layered to promote modularity, separation of concerns, and independent evolution of different aspects of the protocol.

The architecture is divided into three main layers: **HTTP Transport**, **Data Representation**, and **Resource Definition**. Each layer defines specific aspects of the protocol. This layered approach allows for clear separation of concerns.

Encapsulation and **Data Structure** are sub-layers of Data Representation and described later in this document. They focus on the encapsulation and structure of RPP messages, respectively.

Similarily **Data Elements**, their **Mapping** onto Data Structure and **Operations** are elements of Resource Definition. They focus on the semantic structure of RPP resources and transformation of those resources.

```
    +---------------------------------------------------------+
    |                      HTTP Transport                     |
    |                                                         |
    | +-----------------------------------------------------+ |
    | |                 Data Representation                 | |
    | |                                                     | |
    | | +- - - - - - - - - - - - - - - - - - - - - - - - -+ | |
    | | |                  Encapsulation                  | | |
    | |                                                     | |
    | | | +- - - - - - - - - - - - - - - - - - - - - - -+ | | |
    | |   |                Data Structure               |   | |
    | | |                                                 | | |
    | |   | +-----------------------------------------+ |   | |
    | | |   |           Resource Description          |   | | |
    | |   | |                                         | |   | |
    | | |   | +--------------+       +--------------+ | <--------+
    | |   | | |              |       |              | | |   | |  |
    | | |   | |     Data     |       |   Mapping    | |   | | |  |
    | |   | | |   Elements   -------->              -------------+
    | | |   | |              |       |              | |   | | |
    | |   | | |              |       |              | | |   | |
    | | |   | +---^----------+       +--------------+ |   | | |
    | |   | |     |                                   | |   | |
    | | |   |     |                                   |   | | |
    | |   | |     |      +--------------+             | |   | |
    | | |   |     |      |              |             |   | | |
    | |   | |     |      |  Operations  |             | |   | |
    | | |   |     +-------              |             |   | | |
    | |   | |            |              |             | |   | |
    | | |   |            +--------------+             |   | | |
    | |   | |                                         | |   | |
    | | |   +-----------------------------------------+ | | | |
    | |   +- - - - - - - - - - - - - - - - - - - - - - -+   | |
    | | +- - - - - - - - - - - - - - - - - - - - - - - - -+ | |
    | +-----------------------------------------------------+ |
    +---------------------------------------------------------+
```

4.1. Resource Oriented Architecture

RPP adopts a Resource Oriented Architecture (ROA), aligning with RESTful principles. This approach defines all manageable entities as "resources," identified by unique URLs. Operations on these resources are performed through a uniform interface using the standard HTTP methods and their semantics. This contrasts with RPC-style protocols, which often define new and specific operations with custom parameters. ROA promotes a more standardized and interoperable approach, leveraging the existing web infrastructure and its well-defined semantics. Key aspects of ROA within RPP include:

- **Resource Identification:** Each resource is uniquely identifiable by a URL.

- **Uniform Interface:** HTTP methods (HEAD, GET, POST, PUT, DELETE, PATCH) are used to perform operations on resources in a consistent manner.
- **Representation:** Resources can be represented in various formats (e.g., JSON, XML) through HTTP standard content negotiation.
- **Statelessness:** Each request to a resource is treated as independent of previous requests. The server does not maintain client state between requests.
- **Cacheability:** Responses can be cached to improve performance.

4.2. Architecture Layers

4.2.1. HTTP Transport Layer

This layer defines the transport mechanism for RPP messages, utilizing HTTP as the underlying protocol.

It encompasses aspects such as:

- **Authentication and Authorization:** Mechanisms for verifying the identity of clients and controlling access to resources.
- **Resource Addressing using URLs:** Consistent and meaningful URL structures for identifying and accessing resources.
- **Mapping of basic operations to HTTP uniform interface (verbs):** Mapping CRUD (Create, Read, Update, Delete) operations to POST, HEAD/GET, PUT/PATCH, and DELETE respectively.
- **Mapping of operations beyond HTTP uniform interface to URLs and verbs:** Handling more complex operations through appropriate URL structures and HTTP methods.
- **Request routing:** Routing of an RPP request to the correct endpoint based on HTTP attributes and URL structure.
- **RPP specific error codes and relation to HTTP error codes:** Defining RPP-specific error codes while relating them to standard HTTP error codes for consistency.
- **Transaction tracing and idempotency:** Mechanisms for tracking requests and ensuring idempotent operations where appropriate.
- **Caching:** Leveraging HTTP caching mechanisms to improve performance.
- **Content negotiation for media types:** Supporting multiple data representation formats and using content negotiation to select the appropriate format.
- **Language negotiation for textual content:** Supporting multiple languages for textual content and using language negotiation to select the appropriate language.
- **Definition of special resources:** Defining specific resources for service discovery, metadata retrieval, etc.
- **Service discovery mechanisms:** Mechanisms for clients to discover available RPP services.

4.2.2. Data Representation Layer

This layer focuses on the encapsulation and data representation of RPP messages. It defines the media type used to carry RPP data and supports various data representation formats.

It encompasses aspects such as:

- **Data structure:** Defining the structure and schema of the RPP data, potentially using a specific schema language.
- **Data format:** Defining the specific format used to represent RPP data within the representation(e.g., JSON, XML).
- **Data encapsulation:** Optionally defining encapsulation mechanisms for the data format (e.g., JWT, CBOR).
- **Media Type definition:** Defining the specific media type to be used in RPP, including any constraints on the data format and structure

4.2.3. Resource Definition Layer

This layer defines the structure and operations for each resource type, independent of media type or representation. It ensures resources are well-defined and allows for easy extensibility and compatibility with different media types.

It encompasses aspects such as:

- **Data elements:** Defining the individual data elements that make up a resource, including their data types, formats, and any constraints.
- **Resource type definitions:** Defining the structure of specific resource types by combining data elements.
- **IANA registry definitions:** Potentially registering resource definitions with IANA for standardized and automated processing.
- **Mapping of data elements to media types:** Defining how the data elements of a resource type are represented in different media types (e.g., JSON, XML).
- **Extensibility mechanisms on the resource type level:** Providing mechanisms for extending resource types with new data elements or operations.

5. Protocol Details

This section provides further details on each layer of the RPP architecture.

5.1. HTTP Transport Layer Details

The RPP architecture uses the best practices described in [RFC9205] for the HTTP transport layer.

5.1.1. Authentication and Authorization

RPP is aimed to leverage scalable and modern authorization standards, with a focus on OAuth 2.0 [RFC6749] and related frameworks, however it should also support other authentication schemes defined for HTTP, an example would be HTTP Basic Authentication which might be required for compatibility with existing EPP systems. RPP should be able to support future authentication and authorization standards defined for HTTP.

Specifications will define profiles for: * HTTP Authentication schemes (e.g., HTTP Basic Authentication, Bearer Token [RFC6750]) * Authorization frameworks (e.g., OAuth 2.0 [RFC6749])

Implementations will be able to choose authentication and authorization methods appropriate for their security requirements.

5.1.1.1. Authorization Scopes

RPP specifications will standardize authorization scopes to define granular access control for different usage scenarios. These scopes will be defined for various operations and resource types, ensuring that clients can be granted only the necessary permissions.

5.1.1.2. Fine-Grained Authorization

RPP authorization models will be fine-grained, extending beyond simple auth-code based models used EPP. Authorization decisions will be able to consider the specific operation being performed (e.g., update vs. read), the resource being accessed (e.g., a specific domain name), and potentially even attributes within the resource.

Here solutions like OAuth2 RAR [RFC9396] could be considered to provide fine-grained access control.

5.1.2. Resource Addressing

RPP resources are addressed using URLs. Considerations include:

- Hierarchical URL structure to represent resources of different type (e.g., `/domains/{domain-name}`, `/contacts/{contact-id}`).
- URL structure to represent list of related resources (e.g., `/domains/{domain-name}/contacts/`)

RPP URL structure will be designed to be human-readable, intuitive, and RESTful, allowing clients to easily navigate and interact with resources.

RPP would not require all URLs to be hard wired to server's RPP root URL. Instead, it would allow for relative URLs to be defined and discovered by the client. This would allow servers to distibute resources across multiple servers and URLs and allow for easier scaling as described in [RFC9205].

As a matter of extensibility consideration RPP should allow for additional path segments to be added to the URLs and be discoverable by clients.

5.1.2.1. Internationalized Domain Names (IDN)

RPP will address the handling of Internationalized Domain Names (IDNs) in resource addressing. Specifications will define whether to use IDN or UTF-8 encoding directly in URLs and whether to employ redirects to canonical URLs or "see-also" linking for alternative representations. For example, a "see-also" link could point from a UTF-8 encoded URL to an IDN URL and vice versa, allowing clients to use either URL. Another way would be to always redirect to the canonical URL, which would be the IDN URL.

5.1.3. Mapping of basic operations to HTTP uniform interface (verbs)

RPP operations are mapped to standard HTTP methods to leverage the uniform interface and RESTful principles:

- **HEAD:** Retrieve resource state (e.g., retrieving domain existence information) - EPP check command
- **GET:** Retrieve resource state (e.g., retrieving domain or contact information) - EPP info command
- **POST:** Create a new resource (e.g., registering a domain or create contact object) - EPP create command
- **PUT:** Update an existing resource in its entirety (e.g., updating domain registration details) - not 100% equivalent of EPP update command
- **DELETE:** Delete a resource (e.g., deleting a domain registration) - EPP delete command
- **PATCH:** Partially modify a resource (e.g., updating specific attributes of a domain or contact) - EPP update command

EPP transfer commands (query and transform), being in fact a representation of a running process, may be modelled by a subresource `/transfer` of the resource being transferred, with a PUT operation to initiate the transfer, GET operation to query the transfer status and POST operation to approve or reject the transfer. The same approach may apply when adding any other process to the resource, like domain restore.

EPP check command may be modelled either as a GET operation with a dedicated media type, a POST operation with Expect header or a HEAD verb - depending on the specific requirements of the check operation.

Other transform operations like renew, or restore which are not addressable resources in terms of REST may be either also modelled as POST requests with a dedicated media type, or be a convention of URLs with processing resources with only POST interface starting with underscore, e.g. `/domains/{domain-name}/_renew`.

This basic set of rules and guidelines will be further refined in the RPP specifications and give an universal toolset for extending RPP with new resources and commands.

5.1.4. RPP specific error codes and relation to HTTP error codes

RPP utilizes both HTTP status codes and RPP-specific error codes within RPP-specific HTTP Headers and the response body for detailed error reporting and allowing the client or an intermediate to determine what action to take based on status code and header details only.

- Use of HTTP status codes to indicate general categories of errors (e.g., 2xx success responses, 4xx for client errors, 5xx for server errors) [RFC7231].
- Use of additional signalling already standardised for HTTP, for example for rate limiting

- Definition of RPP-specific error codes, warnings of additional processing information, provided in the response, preferably outside of resource representation (e.g. in HTTP Headers) to give granular information about provisioning errors.
- Categorization of RPP error codes as temporary or permanent to guide client retry behavior.

5.1.5. Transaction tracing and idempotency

RPP shall support identification of requests and reponses on both client side and server side with use of client provided identifiers and server provided identifiers. This will allow for tracking of requests and responses in case of errors, and for idempotency of requests. This should be defined outside of the Data Representation Layer (e.g. as HTTP Headers), to assure clear separation of resourse representation from performed actions. If possible existing mechanisms of HTTP shall be employed.

5.1.6. Caching

RPP shall benefit from HTTP standard caching mechanisms to enable standard components like proxies and caches to improve performance and reduce load on servers. RPP shall define caching policies for different resources and operations, including cache-control headers and ETag support.

5.1.7. Content negotiation for media types

RPP supports content negotiation to allow clients to specify preferred media types for request and response payloads using the HTTP 'Accept' and 'Content-Type' headers [RFC7231].

- Support for 'application/rpp+json' as the primary media type.
- Potential support for other media types defined in the Data Representation Layer

5.1.7.1. Prefer Header for Response Verbosity

RPP may utilize the HTTP `Prefer` header [RFC7240] with the "return" preference to allow clients to control the verbosity of responses. For example, clients not interested in full resource representations could use `Prefer: return=minimal` to request minimal responses, reducing payload sizes and improving efficiency. The default behavior, without the `Prefer` header, would be to return a full resource representation, similar to object info responses in EPP, especially after compound requests are completed.

5.1.8. Language negotiation for textual content

RPP shall support language negotiation to enable clients to request responses in a preferred language using the HTTP 'Accept-Language' header [RFC7231].

- Server implementations MAY support multiple languages for textual content in responses to provide human-readable localized responses.
- The default language and mechanisms for indicating supported languages will be defined, preferably using HTTP methods, like OPTIONS or HEAD requests.

- application/rpp+json media type may support multi-language representations, especially for witing operations involving user provided content. Other media types may have different mechanisms for language representation.

5.1.9. Definition of special resources

RPP may define special resources for specific purposes:

- Service Discovery endpoints to advertise protocol capabilities and supported features (see Section 5.1.10).
- Metadata endpoints to provide schema information or other protocol-level metadata, potentially including OpenAPI definitions for documentation and code generation.

5.1.10. Service discovery mechanisms

RPP will define mechanisms for service discovery, allowing clients to dynamically discover RPP service endpoints and capabilities, reducing coupling between clients and servers.

- Potential use of well-known URIs (e.g., `/.well-known/rpp-capabilities`) for service discovery.
- Options for advertising supported protocol versions, extensions, available resource types, authentication methods, and supported features.
- It may be considered for RPP to distribute service discovery for each resource type separately for better scalability and management. For example instead of having a single service discovery endpoint for the whole registry on `/.well-known/rpp-capabilities` there might be a separate discovery placed under `/{resource-type}/.well-known/rpp-capabilities` e.g. `/domains/.well-known/rpp-capabilities`.
- Service discovery shall utilize standardised methods, like URI templates [RFC6570] to allow easy navigation of resources and avoid hard-coding of URLs.

5.2. Data Representation Layer

This layer focuses on the encapsulation and data representation of RPP messages. It defines the media type used to carry RPP data and supports various data representation formats.

5.2.1. Data structure

RPP will define the overall structure of the message payload carried by the chosen media type. Options for the data structure include:

- **'rpp' Structure:** Defining a new, dedicated data structure specifically for RPP messages. This would be the default in core specifications.
- **'epp' Structure Adaptation:** Reusing the existing EPP XML schemas, to maintain data model compatibility with EPP core objects and simplify mapping from EPP.
- **'vc' Structure Leverage:** Utilizing Verifiable Credentials data structures where appropriate, especially for representing identity or authorization information.

5.2.2. Data format and encapsulation

The primary format for RPP data represetations shall be JSON, however RPP should be able to be extended to support other formats like XML, JWT, JWT-SD or CBOR.

- **Plain JSON:** Standard JSON format [RFC8259].
- **XML:** Extensible Markup Language [XML] (considered for potential compatibility with EPP).
- **JWT:** JSON data encapsulated within a JSON Web Token [RFC7519] for potential use-cases when verifiable data consistency is required
- **JWT-SD:** JSON data with Selective Disclosure using JWTs [I-D.draft-ietf-oauth-selective-disclosure-jwt] for minimisation of exposed data.
- **CBOR:** Concise Binary Object Representation for specific use cases requiring compact binary encoding.

Some data formats can be represented in mutiple encapsulations, for example JSON data can be represented in plain JSON, JWT or CBOR. Change of encapsulation shall not affect the data structure. This might be beneficial if RPP is to be extended to support different data formats in the future that only require additional properties provided by encapsulation, like signing, encryption or binary representation.

5.2.3. Media Type definition

Together encapsulation and data structure would define the whole media type. So application/rpp+json would be the primary media type with "rpp" payloads in plain json format. application/epp+xml would be epp payload as per [RFC5730]. The Encapsulation and Data Structure can be also othewise combined as far as it is possible to represent the Data Structure in a given encapsulation. For example it would be straightforward to represent "rpp" structure in JWT format and application/rpp+jwt Media Type, but in order to represent epp structure in JWT format it would require first a mapping of epp messages on JSON instead of XML - rendering application/epp+jwt Media Type.

5.3. Resource Definition Layer

Each resource type, no matter if on a top level, being an independent provisioning object, or a subresource, being a part of another resource, shall be well defined including data elements and possible operations. A respource definition shall on the first level of abstraction be composable out of data elements, without any reference to the media type or representation. This will allow for easy extensibility and compatibility with different media types.

All resource types shall be defined in IANA registry in a way that allows fully automated processing of the resource definition, including data elements, operations and media type representation.

5.3.1. Data Elements

This part defines logical data elements for each resource type, which can also be re-used across resource types. It is abstracted from the actual transport and media type, focusing on the structure and constraints of data elements. Data element definition includes:

- Identification of logical data units (e.g. a stable identifier of a data element, which is independent of the representation)
- Definition of logical data units (e.g., domain name, contact details)
- Format and schema for primitive data elements or reference to other resource type definitions
- Constraints on data elements (e.g., data type, length, allowed values)
- Mechanisms for extensibility, if applicable

Data elements shall be defined in IANA registry in a way that allows for automated processing of the data element definition, including constraints and references to other data elements.

5.3.2. Mapping

This layer defines the mapping of Data Elements onto the Data Representation Layer. For example in case of application/rpp+json media type, the mapping layer would define how the logical data units are represented in JSON format.

This additional level of indirection would allow usage of data formats defined outside of rpp specifications - for example usage of Verifiable Credentials or Verifiable Presentations as first class resource types for contacts in RPP, and mapping appropriate data elements.

The mapping layer shall be defined in IANA registry in a way that allows for automated processing of the mapping definition, including reading and writing operations. Mechanisms, such as defined for JavaScript Object Notation (JSON) Patch [RFC6902], may be used to define the mapping.

5.3.3. Operations

Each resource type shall define operations possible on this resource type. This may encompass any of the mechanism defined on the HTTP transport layer and be constrained by those extensibility rules.

Operations shall be defined in IANA registry in a way that allows for automated processing of the operation definition, including constraints and references to other resource types.

FIXME: find an appropriate section for this * Compatibility Profiles - to define subsets of RPP for specific use cases or EPP compatibility.

6. References

6.1. Informational References

[RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", IETF, STD 69, DOI 10.17487/RFC5730, BCP 69, RFC 5730, August 2009, <https://www.rfc-editor.org/info/rfc5730>.

[RFC5731] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", IETF, STD 69, DOI 10.17487/RFC5731, BCP 69, RFC 5731, August 2009, <https://www.rfc-editor.org/info/rfc5731>.

[RFC5732] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Host Mapping", IETF, STD 69, DOI 10.17487/RFC5732, BCP 69, RFC 5732, August 2009, <https://www.rfc-editor.org/info/rfc5732>.

[RFC5733] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Contact Mapping", IETF, STD 69, DOI 10.17487/RFC5733, BCP 69, RFC 5733, August 2009, <https://www.rfc-editor.org/info/rfc5733>.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", IETF, DOI 10.17487/RFC7231, RFC 7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[REST] "Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Doctoral Dissertation, University of California, Irvine, September 2000, >.".

[RFC7240] Snell, J., "Prefer Header for HTTP", IETF, DOI 10.17487/RFC7240, RFC 7240, June 2014, <https://www.rfc-editor.org/info/rfc7240>.

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", IETF, STD 90, DOI 10.17487/RFC8259, BCP 90, RFC 8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", IETF, DOI 10.17487/RFC6570, RFC 6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", IETF, DOI 10.17487/RFC6749, RFC 6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", IETF, DOI 10.17487/RFC6750, RFC 6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", IETF, DOI 10.17487/RFC7519, RFC 7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC9082] Hollenbeck, S. and A. Newton, "Registration Data Access Protocol (RDAP) Query Format", IETF, STD 95, DOI 10.17487/RFC9082, BCP 95, RFC 9082, June 2021, <https://www.rfc-editor.org/info/rfc9082>.

[RFC6902] Bryan, P. and M. Nottingham, "JavaScript Object Notation (JSON) Patch", IETF, DOI 10.17487/RFC6902, RFC 6902, April 2013, <https://www.rfc-editor.org/info/rfc6902>.

[XML] "Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. and Yergeau, F., "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, []().".

[I-D.draft-ietf-oauth-selective-disclosure-jwt] "Fett D., Yasuda K. and Campbell B. , "Selective Disclosure for JWTs (SD-JWT)", Work in Progress, Internet-Draft, draft-ietf-oauth-selective-disclosure-jwt, 16 January 2025 >".

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", IETF, DOI 10.17487/RFC9396, RFC 9396, May 2023, <https://www.rfc-editor.org/info/rfc9396>.

[RFC9205] Nottingham, M., "Building Protocols with HTTP", IETF, DOI 10.17487/RFC9205, BCP 56, RFC 9205, June 2022, <https://www.rfc-editor.org/info/rfc9205>.

[RPPReq] ""RPP Requirements (Work in progress 7.3.2025)", https:/ /github.com/ietf/wiki.ietf.org/blob/157294ff0fdfb2715da5a287dfba6c641a1bad67/group/rpp/requirements.md".

Authors' Addresses

P Kowalik
DENIC eG
Theodor-Stern-Kai 1
Frankfurt am Main
Germany
Email: pawel.kowalik@denic.de
URI: https://denic.de

M Wullink
SIDN Labs
Netherlands
Email: maarten.wullink@sidn.nl
URI: https://sidn.nl/