Stream: Internet Engineering Task Force (IETF)
RFC: 9560
Category: Standards Track
Published: April 2024
ISSN: 2070-1721
Author: S. Hollenbeck
Verisign Labs

RFC 9560
Federated Authentication for the Registration Data Access Protocol (RDAP) Using OpenID Connect

Abstract

The Registration Data Access Protocol (RDAP) provides Representational State Transfer (RESTful) web services to retrieve registration metadata from domain name and regional internet registries. RDAP allows a server to make access control decisions based on client identity, and as such, it includes support for client identification features provided by the Hypertext Transfer Protocol (HTTP). Identification methods that require clients to obtain and manage credentials from every RDAP server operator present management challenges for both clients and servers, whereas a federated authentication system would make it easier to operate and use RDAP without the need to maintain server-specific client credentials. This document describes a federated authentication system for RDAP based on OpenID Connect.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9560.

Copyright Notice

Table of Contents

1. Introduction

The Registration Data Access Protocol (RDAP) provides Representational State Transfer (RESTful) web services to retrieve registration metadata from domain name and regional internet registries. RDAP allows a server to make access control decisions based on client identity, and as such, it includes support for client identification features provided by the Hypertext Transfer Protocol (HTTP) [RFC9110].

RDAP is specified in multiple documents, including "HTTP Usage in the Registration Data Access Protocol (RDAP)" [RFC7480], "Security Services for the Registration Data Access Protocol (RDAP)" [RFC7481], "Registration Data Access Protocol (RDAP) Query Format" [RFC9082], and "JSON Responses for the Registration Data Access Protocol (RDAP)" [RFC9083]. [RFC7481] describes client identification and authentication services that can be used with RDAP, but it does not specify how any of these services can (or should) be used with RDAP.

1.1. Problem Statement

The conventional "username and password" authentication method does not scale well in the RDAP ecosystem. Assuming that all domain name and address registries will eventually provide RDAP service, it is impractical and inefficient for users to secure login credentials from the hundreds of different server operators. Authentication methods based on usernames and passwords do not provide information that describes the user in sufficient detail (while protecting the personal privacy of the user) for server operators to make fine-grained access control decisions based on the user's identity. The authentication system used for RDAP needs to address all of these needs.

1.2. Approach

A basic level of RDAP service can be provided to users who possess an identifier issued by a recognized provider who can authenticate and validate the user. For example, the identifiers issued by social media services can be used. Users who require higher levels of service (and who are willing to share more information about themselves to gain access to that service) can secure identifiers from specialized providers who are or will be able to provide more detailed information about the user. Server operators can then make access control decisions based on the identification information provided by the user.

A federated authentication system in which an RDAP server outsources identification and authentication services to a trusted identity provider would make it easier to operate and use RDAP by reusing existing identifiers to provide a basic level of access. It can also provide the ability to collect additional user identification information, and that information can be shared with the RDAP server operator with the consent of the user in order to help the server operator make access control decisions. This type of system allows an RDAP server to make access control decisions based on the nature of a query and the identity, authentication, and authorization information that is received from the identity provider. This document describes a federated authentication system for RDAP based on OpenID Connect [OIDC] that meets these needs.

2. Conventions Used in This Document

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

All of the HTTP requests described in this document that are sent from an RDAP client to an RDAP server use the HTTP GET method as specified in [RFC9110].

Long lines in examples are wrapped using "The Single Backslash Strategy" described in [RFC8792].

3. Federated Authentication for RDAP

RDAP itself does not include built-in security services. Instead, RDAP relies on features that are available in other protocol layers to provide needed security services including access control, authentication, authorization, availability, data confidentiality, data integrity, and identification. A description of each of these security services can be found in "Internet Security Glossary, Version 2" [RFC4949]. This document focuses on a federated authentication system for RDAP that provides services for authentication, authorization, and identification, allowing a server operator to make access control decisions. Section 3 of [RFC7481] describes general considerations for RDAP access control, authentication, and authorization.

The conventional client-server authentication model requires clients to maintain distinct credentials for every RDAP server. This situation can become unwieldy as the number of RDAP servers increases. Federated authentication mechanisms allow clients to use one credential to access multiple RDAP servers and reduce client credential management complexity.

3.1. RDAP and OpenID Connect

OpenID Connect 1.0 [OIDCC] is a decentralized, Single Sign-On (SSO) federated authentication system that allows users to access multiple web resources with one identifier instead of having to create multiple server-specific identifiers. Users acquire identifiers from OpenID Providers (OPs).

Relying Parties (RPs) are applications (such as RDAP) that outsource their user authentication function to an OP. OpenID Connect is built on top of the authorization framework provided by the OAuth 2.0 protocol [RFC6749].

The OAuth authorization framework describes a method for users to access protected web resources without having to hand out their credentials. Instead, clients are issued Access Tokens by OpenID Providers with the permission of the resource owners. Using OpenID Connect and OAuth, multiple RDAP servers can form a federation, and clients can access any server in the federation by providing one credential registered with any OP in that federation. The OAuth authorization framework is designed for use with HTTP and thus can be used with RDAP.

3.1.1. Terminology

This document uses the terms "client" and "server" as defined by RDAP [RFC7480].

This document uses the terms "Access Token", "Authorization Code", "Authorization Endpoint", "Authorization Grant", "Client Authentication", "Client Identifier", "Protected Resource", "Refresh Token", "Resource Owner", "Resource Server", and "Token Endpoint" defined by OAuth 2.0 [RFC6749]; the terms "Claim Name", "Claim Value", and "JSON Web Token (JWT)" defined by JSON Web Token (JWT) [RFC7519]; the terms "ID Token" and "UserInfo Endpoint" defined by OpenID Connect Core 1.0 [OIDCC]; and the term "JWT Access Token" defined by [RFC9068]. Additional terms from Section 1.2 of the OpenID Connect Core specification are incorporated by reference.

This document uses the terms "remote" and "default" to describe the relationship between an RDAP server and the OpenID Providers that it interacts with. A "remote" OpenID Provider is one that is identified by the RDAP Client by providing either an Issuer Identifier or an End-User Identifier in a login request. Whether an Issuer Identifier or End-User Identifier can be provided in the login request for the purposes of selecting an OpenID Provider can be determined by retrieving the RDAP Server's OIDC configuration details (see Section 4.1). A "default" OpenID Provider is one that the RDAP Server will use when the RDAP Client does not provide an Issuer Identifier or an End-User Identifier in the login request.

This document uses the term "session" to describe a set of interactions between an RDAP client and an RDAP server during a given period of time. For session-oriented clients (see Section 3.1.2), the RDAP session is a typical HTTP session starting with a farv1_session/login request and ending with either a farv1_session/logout request (see Section 5 for a description of both path segments) or a timeout. For token-oriented clients (see Sections 3.1.2 and 6), the RDAP session corresponds to the lifespan of an authorization obtained from an OP and the corresponding Access Token, including any refreshed Access Tokens.

3.1.2. Client Considerations

Clients that delegate OIDC Authentication to an RDAP server as part of session-oriented interactions and can accept and process HTTP cookies [RFC6265] to maintain the session are known as "session-oriented" clients. This type of RDAP client performs the role of a user agent [RFC9110]. An RDAP server performs the role of an OpenID Connect Core Relying Party (RP). A web browser used to send queries directly to an RDAP server is an example of a session-oriented client. Specifications for this type of client can be found in Section 5.

Clients that perform OIDC Authentication directly, taking the role of an RP in interactions with an OP and sending Access Tokens [RFC6749] to an RDAP server to authorize RDAP queries, are known as "token-oriented" clients. An RDAP server performs resource server [RFC6749] functions to verify the tokens received from the client and RP functions to retrieve information from the OP as necessary to make access control decisions. A web browser running JavaScript received from a web service that sends queries to an RDAP server directly or through its back-end web service is an example of a token-oriented client. Specifications for this type of client can be found in Section 6.

Clients **MAY** operate as either session-oriented or token-oriented clients, but they **MUST** do so consistently by not mixing token-oriented and session-oriented requests while interacting with an OP. Servers **SHOULD** support both types of client to maximize interoperability but **MAY** choose to support only one type of client as required by local policy or operating conditions. A server that does not support a particular client type will not support the protocol features (the data structures, path segments, parameters, and interactions) specified for that client type. Server signaling of supported client types is described in Section 4.1.

3.1.3. Overview

At a high level, RDAP authentication of a session-oriented client using OpenID Connect requires completion of the following steps:

1. An RDAP client sends an RDAP "help" query to an RDAP server to determine the type and capabilities of the OpenID Providers that are used by the RDAP server. This information is returned in the rdapConformance section of the response. A value of "farv1" indicates support for the extension described in this specification. If one or more remote OpenID Providers are supported, the RDAP client **SHOULD** evaluate the additional information described in Section 4.1 in order to discover the capabilities of the RDAP server and optionally obtain the set of supported OPs unless that information is available from a trusted out-of-band source and has already been processed.
2. An RDAP client sends an RDAP "login" request to an RDAP server as described in Section 5.2.
3. The RDAP server prepares an Authentication Request containing the desired request parameters.
4. The RDAP server sends an Authentication Request to an OpenID Provider (OP) Authorization Endpoint and redirects the RDAP client to the OpenID Provider using an HTTP redirect.
5. The OpenID Provider authenticates the End-User.
6. The OpenID Provider obtains End-User consent/authorization.
7. The OpenID Provider sends the RDAP Client back to the RDAP server with an Authorization Code using an HTTP redirect.
8. The RDAP server requests tokens using the Authorization Code at the OpenID Provider's Token Endpoint.
9. The RDAP server receives a response that contains an ID Token and Access Token in the response body.
10. The RDAP server validates the tokens as described in [OIDCC] and retrieves the claims associated with the End-User's identity from the OpenID Provider's UserInfo Endpoint.

The steps above can be described in a sequence diagram:

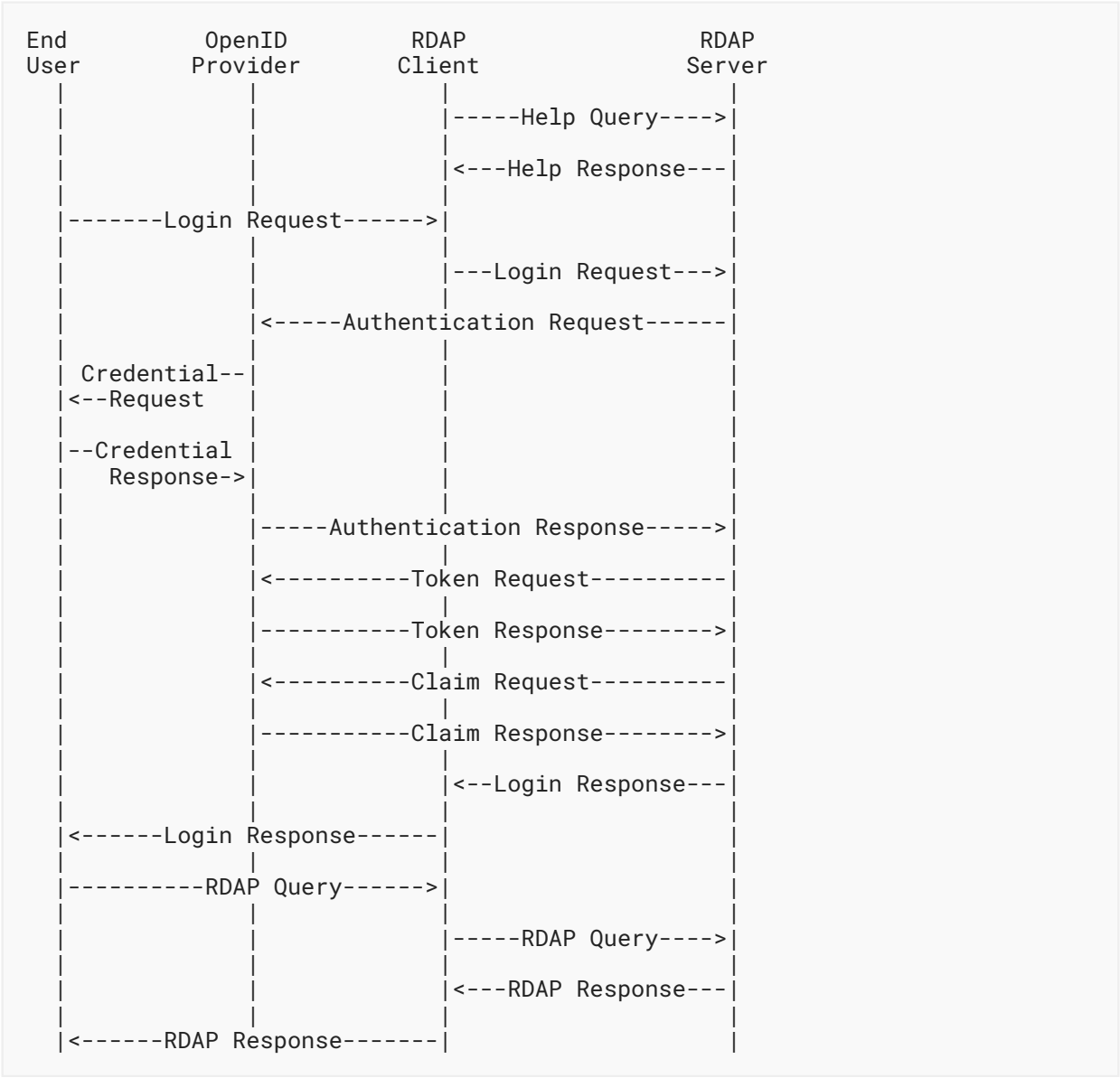

Figure 1

The RDAP server can then make identification, authorization, and access control decisions based on End-User identity information and local policies. Note that OpenID Connect describes different process flows for other types of clients, such as script-based or command-line clients.

RDAP authentication of a token-oriented client using OpenID Connect requires completion of the following steps:

1. An RDAP client sends an RDAP "help" query to an RDAP server to determine the type and capabilities of the OpenID Providers (OPs) that are used by the RDAP server. This information is returned in the rdapConformance section of the response. A value of "farv1" indicates support for the extension described in this specification. If one or more remote OpenID Providers are supported, the RDAP client **SHOULD** evaluate the additional information described in Section 4.1 in order to discover the capabilities of the RDAP server and optionally obtain the set of supported OPs. Support for token-oriented clients requires a default OP.
2. The RDAP client determines the End-User's OP and confirms that it's supported by the RDAP server.
3. The RDAP client sends an Authentication Request to the OP's Authorization Endpoint.
4. The OP authenticates the End-User.
5. The OP obtains End-User consent/authorization.
6. The OP returns an Authorization Code to the RDAP client.
7. The RDAP client requests tokens using the Authorization Code at the OP's Token Endpoint.
8. The RDAP client receives a response that contains an ID Token and an Access Token in the response body.
9. The RDAP client monitors the token validity period and either refreshes the token or requests new tokens as necessary.
10. The RDAP client sends queries that require user identification, authentication, and authorization to an RDAP server that include an Access Token in an HTTP "Authorization" header using the "Bearer" authentication scheme described in [RFC6750].
11. The RDAP server validates the Access Token and retrieves the claims associated with the End-User's identity from the OP's UserInfo Endpoint.
12. The RDAP server determines the End-User's authorization level and processes the query in accordance with server policies.

The steps above can be described in a sequence diagram:

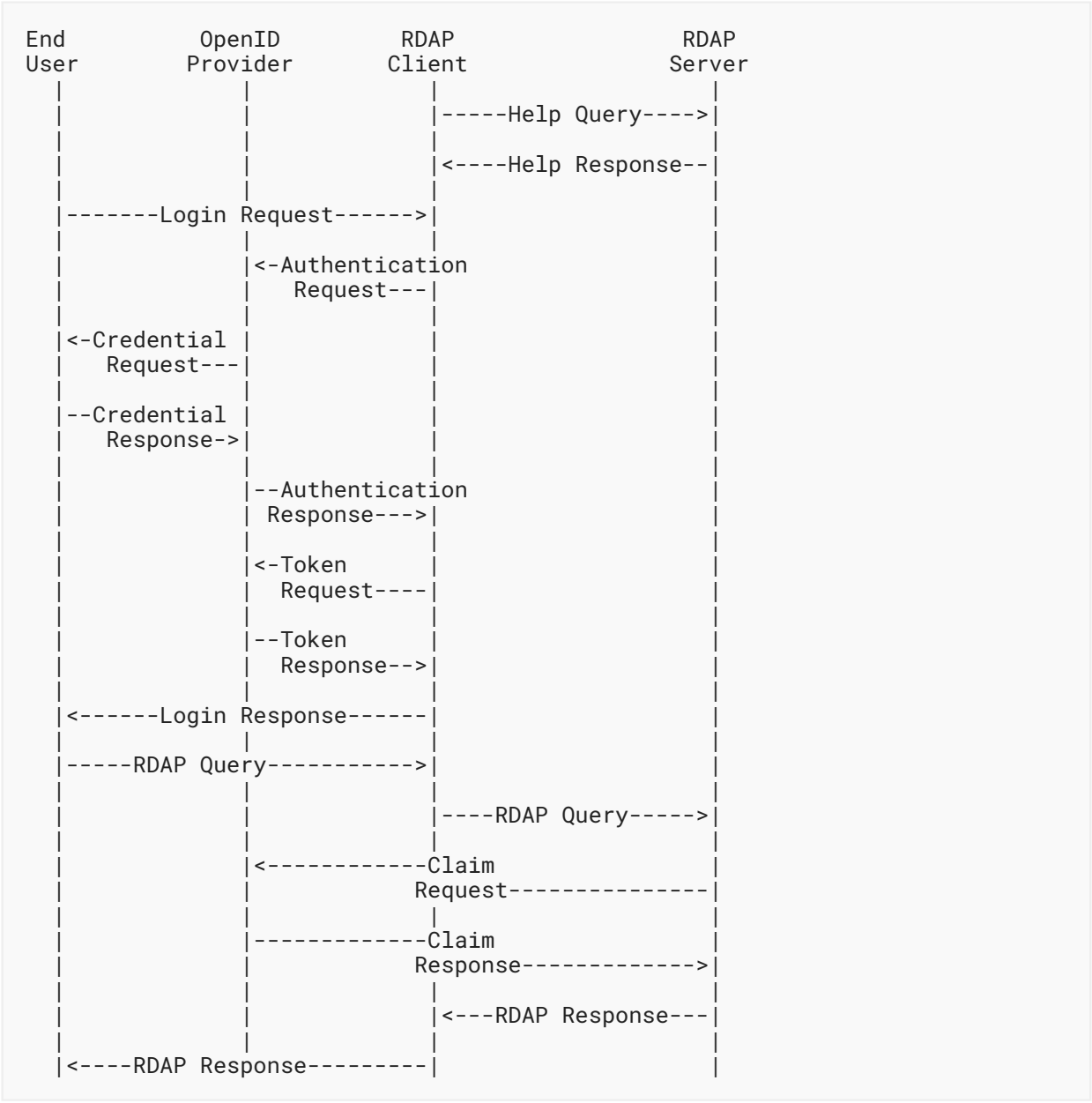

Figure 2

3.1.4. RDAP Authentication and Authorization Steps

End-Users **MAY** present an identifier (an OpenID) issued by an OP to use OpenID Connect with RDAP. If the RDAP server supports a default OpenID Provider or if provider discovery is not supported, the End-User identifier **MAY** be omitted. An OP **SHOULD** include support for the claims described in Section 3.1.5 to provide additional information needed for RDAP End-User authorization; in the absence of these claims, clients and servers **MAY** make authorization and

access control decisions as appropriate given any other information returned from the OP. OpenID Connect requires RPs to register with OPs to use OpenID Connect services for an End-User. The registration process is often completed using out-of-band methods, but it is also possible to use the automated method described by the OpenID Connect Dynamic Client Registration protocol [OIDCR]. The parties involved can use any method that is mutually acceptable.

3.1.4.1. Provider Discovery

An RDAP server/RP needs to be able to map an End-User's identifier to an OP. This can be accomplished using the **OPTIONAL** OpenID Connect Discovery protocol [OIDCD], but that protocol is not widely implemented. Out-of-band methods are also possible and can be more dependable. For example, an RP can support a limited number of OPs and maintain internal associations of those identifiers with the OPs that issued them.

Alternatively, if mapping an End-User's identifier is not possible, or not supported by the RDAP server, the RDAP server **SHOULD** support explicit specification of a remote OP by the RDAP client in the form of a query parameter as described in Section 5.2.2 unless the remote OP has been identified using an out-of-band mechanism. An RDAP server **MUST** provide information about its capabilities and supported OPs in the "help" query response in the "farv1_openidcConfiguration" data structure described in Section 4.1. An RDAP server/RP **MUST** support at least one of these methods of OP discovery.

3.1.4.2. Authentication Request

Once the OP is known, an RP **MUST** form an Authentication Request and send it to the OP as described in Section 3 of the OpenID Connect Core protocol [OIDCC]. The authentication path followed (authorization, implicit, or hybrid) will depend on the Authentication Request response_type set by the RP. The remainder of the processing steps described here assume that the Authorization Code Flow is being used by setting "response_type=code" in the Authentication Request.

The benefits of using the Authorization Code Flow for authenticating a human user are described in Section 3.1 of the OpenID Connect Core protocol [OIDCC]. The Implicit Flow is more commonly used by clients implemented in a web browser using a scripting language; it is described in Section 3.2 of the OpenID Connect Core protocol [OIDCC]. At the time of this writing, the Implicit Flow is considered insecure and efforts are being made to deprecate the flow. The Hybrid Flow (described in Section 3.3 of the OpenID Connect Core protocol [OIDCC]) combines elements of the Authorization Code and Implicit Flows by returning some tokens from the Authorization Endpoint and others from the Token Endpoint.

An Authentication Request can contain several parameters. **REQUIRED** parameters are specified in Section 3.1.2.1 of the OpenID Connect Core protocol [OIDCC]. Apart from these parameters, it is **RECOMMENDED** that the RP include the optional "login_hint" parameter in the request, with the value being that of the "farv1_id" query parameter of the End-User's RDAP "login" request, if provided. Passing the "login_hint" parameter allows a client to pre-fill login form information, so logging in can be more convenient for users. Other parameters **MAY** be included.

The OP receives the Authentication Request and attempts to validate it as described in Section 3.1.2.2 of the OpenID Connect Core protocol [OIDCC]. If the request is valid, the OP attempts to authenticate the End-User as described in Section 3.1.2.3 of the OpenID Connect Core protocol [OIDCC]. The OP returns an error response if the request is not valid or if any error is encountered.

3.1.4.3. End-User Authorization

After the End-User is authenticated, the OP **MUST** obtain consent from the End-User to release authorization information to the RDAP Server/RP. This process is described in Section 3.1.2.4 of the OpenID Connect Core protocol [OIDCC].

3.1.4.4. Authorization Response and Validation

After obtaining an authorization result, the OP will send a response to the RP that provides the result of the authorization process using an Authorization Code. The RP **MUST** validate the response. This process is described in Sections 3.1.2.5 - 3.1.2.7 of the OpenID Connect Core protocol [OIDCC].

3.1.4.5. Token Processing

The RP sends a Token Request using the Authorization Grant to a Token Endpoint to obtain a Token Response containing an Access Token, ID Token, and an **OPTIONAL** Refresh Token. The RP **MUST** validate the Token Response. This process is described in Section 3.1.3.5 of the OpenID Connect Core protocol [OIDCC].

3.1.4.6. Delivery of User Information

The set of claims can be retrieved by sending a request to a UserInfo Endpoint using the Access Token. The claims are returned in the ID Token. The process of retrieving claims from a UserInfo Endpoint is described in Section 5.3 of the OpenID Connect Core protocol [OIDCC].

OpenID Connect specifies a set of standard claims in Section 5.1 of the OpenID Connect Core protocol [OIDCC]. Additional claims for RDAP are described in Section 3.1.5.

3.1.5. Specialized Claims and Authorization Scope for RDAP

OpenID Connect claims are pieces of information used to make assertions about an entity. Section 5 of the OpenID Connect Core protocol [OIDCC] describes a set of standard claims. Section 5.1.2 of [OIDCC] notes that additional claims **MAY** be used, and it describes a method to create them. The set of claims that are specific to RDAP are associated with an OAuth scope request parameter value (see Section 3.3 of [RFC6749]) of "rdap".

3.1.5.1. Stated Purposes

Communities of RDAP users and operators may wish to make and validate claims about a user's "need to know" when it comes to requesting access to a protected resource. For example, a law enforcement agent or a trademark attorney may wish to be able to assert that they have a legal

right to access a protected resource, and a server operator may need to be able to receive and validate that claim. These needs can be met by defining and using an additional "rdap_allowed_purposes" claim.

The "rdap_allowed_purposes" claim identifies the purposes for which access to a protected resource can be requested by an End-User. Use of the "rdap_allowed_purposes" claim is **OPTIONAL**; processing of this claim is subject to server acceptance of the purposes, the trust level assigned to this claim by the server, and successful authentication of the End-User. Unrecognized purpose values **MUST** be ignored, and the associated query **MUST** be processed as if the unrecognized purpose value was not present at all. See Section 9.3 for a description of the IANA considerations associated with this claim.

The "rdap_allowed_purposes" claim is represented as an array of case-sensitive StringOrURI values as specified in Section 2 of the JSON Web Token (JWT) specification [RFC7519]. An example:

"rdap_allowed_purposes": ["domainNameControl","dnsTransparency"]

Purpose values are assigned to an End User's credential by an Identity Provider. Identity Providers **MUST** ensure that appropriate purpose values are only assigned to End User identities that are authorized to use them.

3.1.5.2. Do Not Track

Communities of RDAP users and operators may wish to make and validate claims about a user's wish to not have their queries logged, tracked, or recorded. For example, a law enforcement agent may wish to assert that their queries are part of a criminal investigation and should not be tracked due to a risk of query exposure compromising the investigation, and a server operator may need to be able to receive and validate that claim. These needs can be met by defining and using an additional "do not track" claim.

The "do not track" ("rdap_dnt_allowed") claim can be used to identify an End-User that is authorized to perform queries without the End-User's association with those queries being logged, tracked, or recorded by the server. Client use of the "rdap_dnt_allowed" claim is **OPTIONAL**. Server operators **MUST NOT** log, track, or record any association of the query and the End-User's identity if the End-User is successfully identified and authorized, if the "rdap_dnt_allowed" claim is present, if the value of the claim is "true", and if accepting the claim complies with local regulations regarding logging and tracking.

The "rdap_dnt_allowed" value is represented as a JSON boolean literal. An example:

rdap_dnt_allowed: true

No special query tracking processing is required if this claim is not present or if the value of the claim is "false". Use of this claim **MUST** be limited to End-Users who are granted "do not track" privileges in accordance with service policies and regulations. Specification of these policies and regulations is beyond the scope of this document.

4. Common Protocol Features

As described in Section 3.1.4.1, an RDAP server **MUST** provide information about its capabilities and supported OPs in a "help" query response. This specification describes a new "farv1_openidcConfiguration" data structure that describes the OpenID Connect configuration and related extension features supported by the RDAP server. This data structure is returned to all client types.

4.1. OpenID Connect Configuration

The "farv1_openidcConfiguration" data structure is an object with the following members:

1. "sessionClientSupported": (**REQUIRED**) a boolean value that describes RDAP server support for session-oriented clients (see Section 3.1.2).
2. "tokenClientSupported": (**REQUIRED**) a boolean value that describes RDAP server support for token-oriented clients (see Section 3.1.2).
3. "dntSupported": (**REQUIRED**) a boolean value that describes RDAP server support for the "farv1_dnt" query parameter (see Section 4.2.2).
4. "providerDiscoverySupported": (**OPTIONAL**) a boolean value that describes RDAP server support for discovery of providers of End-User identifiers. The default value is "true".
5. "issuerIdentifierSupported": (**OPTIONAL**) a boolean value that describes RDAP server support for explicit client specification of an Issuer Identifier. The default value is "true".
6. "implicitTokenRefreshSupported": (**OPTIONAL**) a boolean value that describes RDAP server support for implicit token refresh. The default value is "false".
7. "openidcProviders": (**OPTIONAL**) a list of objects with the following members that describes the set of OPs that are supported by the RDAP server. This data is **RECOMMENDED** if the value of issuerIdentifierSupported is "true":
   a. "iss": (**REQUIRED**) a URI value that represents the Issuer Identifier of the OP as per the OpenID Connect Core specification [OIDCC].
   b. "name": (**REQUIRED**) a string value representing the human-friendly name of the OP.
   c. "default": (**OPTIONAL**) a boolean value that describes RDAP server support for an **OPTIONAL** default OP that will be used when a client omits the "farv1_id" and "farv1_iss" query parameters from a "farv1_session/login" request. Only one member of this set can be identified as the default OP by setting a value of "true". The default value is "false".
   d. "additionalAuthorizationQueryParams": (**OPTIONAL**) an object where each member represents an OAuth authorization request parameter name-value pair supported by the OP. The name represents an OAuth query parameter, and the value is the query parameter value. A token-oriented RDAP client **SHOULD** add these query parameters and their corresponding values to the Authentication Request URL when requesting authorization by a specified OP through a proxy OP.

An RDAP server **MUST** set either the "sessionClientSupported" or the "tokenClientSupported" value to "true". Both values **MAY** be set to "true" if an RDAP server supports both types of clients.

The "providerDiscoverySupported" value has a direct impact on the use of the "farv1_id" query parameter described in Sections 3.1.4.2 and 5.2.1. The value of "providerDiscoverySupported" **MUST** be "true" for an RDAP server to properly accept and process "farv1_id" query parameters. Similarly, the "issuerIdentifierSupported" value has a direct impact on the use of the "farv1_iss" query parameter described in Section 5.2.2. The value of "issuerIdentifierSupported" **MUST** be "true" for an RDAP server to properly accept and process "farv1_iss" query parameters.

An example of a "farv1_openidcConfiguration" data structure:

```
"farv1_openidcConfiguration": {
  "sessionClientSupported": true,
  "tokenClientSupported": true,
  "dntSupported": false,
  "providerDiscoverySupported": true,
  "issuerIdentifierSupported": true,
  "openidcProviders":
    [
      {
        "iss": "https://idp.example.com",
        "name": "Example IDP"
      },
      {
        "iss": "https://accounts.example.net",
        "name": "Login with EXAMPLE",
        "additionalAuthorizationQueryParams": {
          "kc_idp_hint": "examplePublicIDP"
        }
      },
      {
        "iss": "https://auth.nic.example/auth/realms/rdap",
        "name": "Default OP for the Example RDAP server",
        "default": true
      }
    ]
}
```

Figure 3

4.2. RDAP Query Parameters

This specification describes two **OPTIONAL** query parameters for use with RDAP queries that request access to information associated with protected resources:

"farv1_qp": A query parameter to identify the purpose of the query.

"farv1_dnt": A query parameter to request that the server not log or otherwise record information about the identity associated with a query.

One or both parameters **MAY** be added to an RDAP request URI using the syntax described in the "application/x-www-form-urlencoded" section of the WHATWG URL Standard [HTMLURL].

4.2.1. RDAP Query Purpose

This query is represented as a "key=value" pair using a key value of "farv1_qp" and a value component that contains a single query purpose string from the set of allowed purposes associated with the End-User's identity (see Section 3.1.5.1). If present, the server **SHOULD** compare the value of the parameter to the "rdap_allowed_purposes" claim values associated with the End-User's identity and ensure that the requested purpose is present in the set of allowed purposes. The RDAP server **MAY** choose to ignore both the requested purpose and the "rdap_allowed_purposes" claim values if they are inconsistent with local server policy. The server **MUST** return an HTTP 403 (Forbidden) response if the requested purpose is not an allowed purpose. If the "farv1_qp" parameter is not present, the server **MUST** process the query and make an access control decision based on any other information known to the server about the End-User and the information they are requesting. For example, a server **MAY** treat the request as one performed by an unidentified or unauthenticated user and return either an error or an appropriate subset of the available data. An example domain query using the "farv1_qp" query parameter:

https://example.com/rdap/domain/example.com?farv1_qp=legalActions

4.2.2. RDAP Do Not Track

This query is represented as a "key=value" pair using a key value of "farv1_dnt" and a value component that contains a single boolean value. A value of "true" indicates that the End-User is requesting that their query is not tracked or logged in accordance with server policy. A value of "false" indicates that the End-User is accepting that their query can be tracked or logged in accordance with server policy. The server **MUST** return an HTTP 403 (Forbidden) response if the server is unable to perform the action requested by this query parameter. An example domain query using the "farv1_dnt" query parameter:

https://example.com/rdap/domain/example.com?farv1_dnt=true

4.2.3. Parameter Processing

Unrecognized query parameters **MUST** be ignored. An RDAP server that processes an authenticated query **MUST** determine if the End-User identification information is associated with an OP that is recognized and supported by the server. RDAP servers **MUST** reject queries that include identification information that is not associated with a supported OP by returning an HTTP 400 (Bad Request) response. An RDAP server that receives a query containing identification information associated with a recognized OP **MUST** perform the steps required to authenticate the user with the OP, process the query, and return an RDAP response that is appropriate for the End-User's level of authorization and access.

5. Protocol Features for Session-Oriented Clients

This specification adds the following features to RDAP that are commonly used by session-oriented clients:

1. Data structures to return information that describes an established session and the information needed to establish a session for a UI-constrained device.
2. A query parameter to request authentication for a specific End-User identity.
3. A query parameter to support authentication for a specific End-User identity on a device with a constrained user interface.
4. A query parameter to identify the purpose of the query.
5. A query parameter to request that the server not log or otherwise record information about the identity associated with a query.
6. Path segments to start, stop, refresh, and determine the status of an authenticated session for a specific End-User identity.

5.1. Data Structures

This specification describes two new data structures that are used to return information to a session-oriented client:

"farv1_session": A data structure that contains information that describes an established session.

"farv1_deviceInfo": A data structure that contains information that describes an active attempt to establish a session on a UI-constrained device.

5.1.1. Session

The "farv1_session" data structure is an object that contains the following members:

1. "userID": an **OPTIONAL** string value that represents the End-User identifier associated with the session.
2. "iss": an **OPTIONAL** URI value that represents the issuer of the End-User identifier associated with the session.
3. "userClaims": an **OPTIONAL** object that contains the set of claims associated with the End-User's identity based on the user information provided by the OP as described in Section 3.1.4.6 and processed by the RDAP server in the authentication and authorization process. The set of possible values is determined by OP policy and RDAP server policy.
4. "sessionInfo": an **OPTIONAL** object that contains two members:
   a. "tokenExpiration": an integer value that represents the number of seconds that remain in the lifetime of the Access Token.
   b. "tokenRefresh": a boolean value that indicates if the OP supports refresh tokens. As described in [RFC6749], support for refresh tokens is **OPTIONAL**.

Note that all of the members of the "farv1_session" data structure are **OPTIONAL**. See Section 5.2.3 for instructions describing when to return the minimum set of members.

An example of a "farv1_session" data structure:

```
"farv1_session": {
  "userID": "user.idp.example",
  "iss": "https://idp.example.com",
  "userClaims": {
    "sub": "103892603076825016132",
    "name": "User Person",
    "given_name": "User",
    "family_name": "Person",
    "picture": "https://lh3.example.com/a-/AOh14=s96-c",
    "email": "user@example.com",
    "email_verified": true,
    "locale": "en",
    "rdap_allowed_purposes": [
      "domainNameControl",
      "personalDataProtection"
    ],
    "rdap_dnt_allowed": false
  },
  "sessionInfo": {
    "tokenExpiration": 3599,
    "tokenRefresh": true
  }
}
```

Figure 4

5.1.2. Device Info

The flow described in Section 3.1.4 requires an End-User to interact with a server using a user interface that can process HTTP. This will not work well in situations where the client is automated or an End-User is using a command-line user interface such as curl or wget. This limitation can be addressed using a web browser on a second device. The information that needs to be entered using the web browser is contained in the "farv1_deviceInfo" data structure, an object that contains members as described in Section 3.2 of [RFC8628].

An example of a "farv1_deviceInfo" data structure:

```
"farv1_deviceInfo": {
  "device_code": "AH-1ng2ezu",
  "user_code": "NJJQ-GJFC",
  "verification_uri": "https://www.example.com/device",
  "verification_uri_complete":
     "https://www.example.com/device?user_code=NJJQ-GJFC",
  "expires_in": 1800,
  "interval": 5
}
```

Figure 5

5.2. Client Login

Client authentication is requested by sending a "farv1_session/login" request to an RDAP server. If the RDAP server supports only remote OpenID Providers, the "farv1_session/login" request **MUST** include at least one End-User Identifier or OP Issuer Identifier.

The server sets an HTTP cookie as described in [RFC6265] when the "farv1_session/login" request is received and processed successfully. The client **MUST** include the session cookie received from the server in any RDAP request within the scope of that session, including "farv1_session/refresh", "farv1_session/status", and "farv1_session/logout". A "farv1_session/login" followed by another "farv1_session/login" that does not include an HTTP cookie **MUST** start a new session on the server that includes a new cookie. A server that receives a "farv1_session/login" followed by another "farv1_session/login" that includes an HTTP cookie **MUST** return an HTTP 409 (Conflict) response.

To help reduce the risk of resource starvation, a server **MAY** reject a "farv1_session/login" request and refuse to start a new session by returning an HTTP 409 (Conflict) response if a server-side maximum number of concurrent sessions per user exists and the client exceeds that limit. Additionally, an active session **MAY** be removed by the server due to timeout expiration or because a maximum session lifetime has been exceeded. Clients **SHOULD** proactively monitor the "tokenExpiration" value associated with an active session and refresh the session as appropriate to provide a positive user experience.

5.2.1. End-User Identifier

The End-User identifier is delivered using one of two methods: by adding a query component to an RDAP request URI using the syntax described in the "application/x-www-form-urlencoded" section of the WHATWG URL Standard [HTMLURL] or by including an HTTP "Authorization" request header for the Basic authentication scheme as described in [RFC7617]. Clients can use either of these methods to deliver the End-User identifier to a server that supports remote OpenID Providers and provider discovery. Servers that support remote OpenID Providers and provider discovery **MUST** accept both methods. If the RDAP server supports a default OpenID Provider or if provider discovery is not supported, the End-User identifier **MAY** be omitted.

The query parameter used to deliver the End-User identifier is represented as an **OPTIONAL** "key=value" pair using a key value of "farv1_id" and a value component that contains the client identifier issued by an OP. An example for client identifier "user.idp.example":

```
========== NOTE: '\' line wrapping per RFC 8792 ===========
https://example.com/rdap/farv1_session/\
login?farv1_id=user.idp.example
```

The authorization header for the Basic authentication scheme contains a base64-encoded representation of the client identifier issued by an OP. No password is provided. An example for client identifier "user.idp.example":

https://example.com/rdap/farv1_session/login

Authorization: Basic dXNlci5pZHAuZXhhbXBsZQ==

An example for use with a default OpenID Provider:

https://example.com/rdap/farv1_session/login

5.2.2. OP Issuer Identifier

The OP's Issuer Identifier is delivered by adding a query component to an RDAP request URI using the syntax described in the "application/x-www-form-urlencoded" section of the WHATWG URL Standard [HTMLURL]. If the RDAP server supports a default OpenID Provider, the Issuer Identifier **MAY** be omitted.

The query parameter used to deliver the OP's Issuer Identifier is represented as an **OPTIONAL** "key=value" pair using a key value of "farv1_iss" and a value component that contains the Issuer Identifier associated with an OP. An RDAP server **MAY** accept Issuer Identifiers not specified in the "farv1_openidcConfiguration" data structure and **MAY** also decide to accept specific Issuer Identifiers only from specific clients. An example for Issuer Identifier "https://idp.example.com":

```
========== NOTE: '\' line wrapping per RFC 8792 ===========
https://example.com/rdap/farv1_session/\
login?farv1_iss=https://idp.example.com
```

5.2.3. Login Response

The response to this request **MUST** be a valid RDAP response per [RFC9083]. It **MUST NOT** include any members that relate to a specific RDAP object type (e.g., "events" or "status"). In addition, the response **MAY** include an indication of the requested operation's success or failure in the "notices" data structure. If successful, the response **MUST** include a "farv1_session" data structure that includes a "sessionInfo" object and an **OPTIONAL** "userClaims" object. If unsuccessful, the response **MUST** include a "farv1_session" data structure that omits the "userClaims" and "sessionInfo" objects.

An example of a successful "farv1_session/login" response:

```
{
  "rdapConformance": [
    "farv1"
  ],
  "lang": "en-US",
  "notices": [
    {
      "title": "Login Result",
      "description": [
        "Login succeeded"
      ]
    }
  ],
  "farv1_session": {
    "userID": "user.idp.example",
    "iss": "https://idp.example.com",
    "userClaims": {
      "sub": "103892603076825016132",
      "name": "User Person",
      "given_name": "User",
      "family_name": "Person",
      "picture": "https://lh3.example.com/a-/AOh14=s96-c",
      "email": "user@example.com",
      "email_verified": true,
      "locale": "en",
      "rdap_allowed_purposes": [
        "domainNameControl",
        "personalDataProtection"
      ],
      "rdap_dnt_allowed": false
    },
    "sessionInfo": {
      "tokenExpiration": 3599,
      "tokenRefresh": true
    }
  }
}
```

Figure 6

An example of a failed "farv1_session/login" response:

```
{
  "rdapConformance": [
    "farv1"
  ],
  "lang": "en-US",
  "notices": [
    {
      "title": "Login Result",
      "description": [
        "Login failed"
      ]
    }
  ],
  "farv1_session": {
    "userID": "user.idp.example",
    "iss": "https://idp.example.com"
  }
}
```

Figure 7

5.2.4. Clients with Limited User Interfaces

"OAuth 2.0 Device Authorization Grant" [RFC8628] provides an **OPTIONAL** method to request user authorization from devices that have an Internet connection but lack a suitable browser for a more conventional OAuth flow. This method requires an End-User to use a second device (such as a smart telephone) that has access to a web browser for entry of a code sequence that is presented on the UI-constrained device.

5.2.4.1. UI-Constrained Client Login

Client authentication is requested by sending a "farv1_session/device" request to an RDAP server. If the RDAP server supports only remote OpenID Providers, the "farv1_session/device" request **MUST** include either an End-User identifier as described in Section 5.2.1 or an OP Issuer Identifier as described in Section 5.2.2.

An example using wget for client identifier "user.idp.example":

```
========== NOTE: '\' line wrapping per RFC 8792 ===========
   wget -qO- "https://example.com/rdap/farv1_session/device\
   ?farv1_id=user.idp.example"
```

Figure 8

The authorization header for the Basic authentication scheme contains a base64-encoded representation of the client identifier issued by an OP. No password is provided.

An example using curl and an authorization header:

```
========== NOTE: '\' line wrapping per RFC 8792 ===========
   curl -H "Authorization: Basic dXNlci5pZHAuZXhhbXBsZQ==\"
   "https://example.com/rdap/farv1_session/device"
```

Figure 9

The response to this request **MUST** be a valid RDAP response per [RFC9083]. It **MUST NOT** include any members that relate to a specific RDAP object type (e.g., "events" or "status"). In addition, the response **MAY** include an indication of the requested operation's success or failure in the "notices" data structure and, if successful, a "farv1_deviceInfo" data structure.

An example of a "farv1_session/device" response:

```
  {
    "rdapConformance": [
      "farv1"
    ],
    "lang": "en-US",
    "notices": [
      {
        "title": "Device Login Result",
        "description": [
          "Login succeeded"
        ]
      }
    ],
    "farv1_deviceInfo": {
      "device_code": "AH-1ng2ezu",
      "user_code": "NJJQ-GJFC",
      "verification_uri": "https://www.example.com/device",
      "verification_uri_complete":
              "https://www.example.com/device?user_code=NJJQ-GJFC",
      "expires_in": 1800,
          "interval": 5
    }
  }
```

Figure 10

5.2.4.2. UI-Constrained Client Login Polling

After successful processing of the "farv1_session/device" request, the client **MUST** send a "farv1_session/devicepoll" request to the RDAP server to continue the login process. This request initiates the polling function described in [RFC8628] on the RDAP server. The RDAP server polls the OP as described in Section 3.4 of [RFC8628], allowing the RDAP server to wait for the End-User to enter the information returned from the "farv1_session/device" request using the interface on their second device. After the End-User has completed that process, or if the process

fails or times out, the OP will respond to the polling requests with an indication of success or failure. If the RDAP server supports only remote OpenID Providers, the "farv1_session/devicepoll" request **MUST** include either an End-User identifier as described in Section 5.2.1 or an OP Issuer Identifier as described in Section 5.2.2.

The "farv1_session/devicepoll" request **MUST** also include a "farv1_dc" query parameter. The query parameter is represented as an **OPTIONAL** "key=value" pair using a key value of "farv1_dc" and a value component that contains the value of the device_code that was returned in the response to the "farv1_session/device" request.

An example using wget:

```
========== NOTE: '\' line wrapping per RFC 8792 ===========
   wget -qO- --keep-session-cookies --save-cookies cookie.txt\
   "https://example.com/rdap/farv1_session/devicepoll\
   ?farv1_id=user.idp.example&farv1_dc=AH-1ng2ezu"
```

Figure 11

An example using curl:

```
========== NOTE: '\' line wrapping per RFC 8792 ===========
   curl -c cookie.txt "https://example.com/rdap/farv1_session/\
   devicepoll?farv1_id=user.idp.example&farv1_dc=AH-1ng2ezu"
```

Figure 12

The response to this request **MUST** use the response structures described in Section 5.2. RDAP query processing can continue normally on the UI-constrained device once the device polling process has been completed successfully.

5.3. Session Status

Clients **MAY** send a query to an RDAP server to determine the status of an existing login session using a "farv1_session/status" path segment. An example "farv1_session/status" request:

https://example.com/rdap/farv1_session/status

The response to this request **MUST** be a valid RDAP response per [RFC9083]. It **MUST NOT** include any members that relate to a specific RDAP object type (e.g., "events" or "status"). In addition, the response **MAY** include an indication of the requested operation's success or failure in the "notices" data structure. If the operation is successful and an active session exists, the response **MUST** include a "farv1_session" data structure that includes a "sessionInfo" object and an **OPTIONAL** "userClaims" object. If the operation is unsuccessful or if no active session exists, the response **MUST NOT** include a "farv1_session" object.

An example of a "farv1_session/status" response for an active session:

```
{
  "rdapConformance": [
    "farv1"
  ],
  "lang": "en-US",
  "notices": [
    {
      "title": "Session Status Result",
      "description": [
        "Session status succeeded"
      ]
    }
  ],
  "farv1_session": {
    "userID": "user.idp.example",
    "iss": "https://idp.example.com",
    "userClaims": {
      "sub": "103892603076825016132",
      "name": "User Person",
      "given_name": "User",
      "family_name": "Person",
      "picture": "https://lh3.example.com/a-/AOh14=s96-c",
      "email": "user@example.com",
      "email_verified": true,
      "locale": "en",
      "rdap_allowed_purposes": [
        "domainNameControl",
        "personalDataProtection"
      ],
      "rdap_dnt_allowed": false
    },
    "sessionInfo": {
      "tokenExpiration": 3490,
      "tokenRefresh": true
    }
  }
}
```

Figure 13

If the operation is successful and an active session does not exist, the response **MAY** note the lack of an active session in the "notices" data structure. The "farv1_session" data structure **MUST** be omitted.

An example of a "farv1_session/status" response with no active session:

```
{
  "rdapConformance": [
    "farv1"
  ],
  "lang": "en-US",
  "notices": [
    {
      "title": "Session Status Result",
      "description": [
        "Session status succeeded",
        "No active session"
      ]
    }
  ]
}
```

Figure 14

5.4. Session Refresh

Clients **MAY** send a request to an RDAP server to refresh or extend an existing login session using a "farv1_session/refresh" path segment. The RDAP server **MAY** attempt to refresh the Access Token associated with the current session as part of extending the session for a period of time determined by the RDAP server. As described in [RFC6749], OP support for refresh tokens is **OPTIONAL**. An RDAP server **MUST** determine if the OP supports token refresh and process the refresh request by either requesting refresh of the Access Token or returning a response that indicates that token refresh is not supported by the OP in the "notices" data structure. An example "farv1_session/refresh" request:

https://example.com/rdap/farv1_session/refresh

The response to this request **MUST** be a valid RDAP response per [RFC9083]. It **MUST NOT** include any members that relate to a specific RDAP object type (e.g., "events" or "status"). In addition, the response **MAY** include an indication of the requested operation's success or failure in the "notices" data structure. The response **MUST** include a "farv1_session" data structure that includes a "sessionInfo" object and an **OPTIONAL** "userClaims" object. If unsuccessful but an active session exists, the response **MUST** include a "farv1_session" data structure that includes a "sessionInfo" object and an **OPTIONAL** "userClaims" object. If unsuccessful and no active session exists, the response **MUST** omit the "farv1_session" data structure.

An example of a successful "farv1_session/refresh" response:

```
{
  "rdapConformance": [
    "farv1"
  ],
  "lang": "en-US",
  "notices": [
    {
      "title": "Session Refresh Result",
      "description": [
        "Session refresh succeeded",
        "Token refresh succeeded."
      ]
    }
  ],
  "farv1_session": {
    "userID": "user.idp.example",
    "iss": "https://idp.example.com",
    "userClaims": {
      "sub": "103892603076825016132",
      "name": "User Person",
      "given_name": "User",
      "family_name": "Person",
      "picture": "https://lh3.example.com/a-/AOh14=s96-c",
      "email": "user@example.com",
      "email_verified": true,
      "locale": "en",
      "rdap_allowed_purposes": [
        "domainNameControl",
        "personalDataProtection"
      ],
      "rdap_dnt_allowed": false
    },
    "sessionInfo": {
      "tokenExpiration": 3599,
      "tokenRefresh": true
    }
  }
}
```

Figure 15

Alternatively, an RDAP server **MAY** attempt to refresh an Access Token upon receipt of a query if the Access Token associated with an existing session has expired and the corresponding OP supports token refresh. The default RDAP server behavior is described in the "implicitTokenRefreshSupported" value that's included in the "farv1_openidcConfiguration" data structure (see Section 4.1).

If the value of "implicitTokenRefreshSupported" is "true", the client **MAY** either explicitly attempt to refresh the session using the "farv1_session/refresh" query or depend on the RDAP server to attempt to refresh the session as necessary when an RDAP query is received by the server. In this case, a server **MUST** attempt to refresh the Access Token upon receipt of a query if the Access

Token associated with an existing session has expired and the corresponding OP supports token refresh. Servers **MUST** return an HTTP 401 (Unauthorized) response to a query if an attempt to implicitly refresh an existing session fails.

If the value of "implicitTokenRefreshSupported" is "false", the client **MUST** explicitly attempt to refresh the session using the "farv1_session/refresh" query to extend an existing session. If a session cannot be extended for any reason, the client **MUST** establish a new session to continue authenticated query processing by submitting a "farv1_session/login" query. If the OP does not support token refresh, the client **MUST** submit a new "farv1_session/login" request to establish a new session once an Access Token has expired.

Clients **SHOULD NOT** send a "farv1_session/refresh" request in the absence of an active login session because the request conflicts with the current state of the server. Servers **MUST** return an HTTP 409 (Conflict) response if a "farv1_session/refresh" request is received in the absence of a session cookie.

5.5. Client Logout

Clients **MAY** send a request to an RDAP server to terminate an existing login session. Termination of a session is requested using a "farv1_session/logout" path segment. Access and refresh tokens can be revoked during the "farv1_session/logout" process as described in [RFC7009] if supported by the OP (token revocation endpoint support is **OPTIONAL** per [RFC8414]). If supported, this feature **SHOULD** be used to ensure that the tokens are not mistakenly associated with a future RDAP session. Alternatively, an RDAP server **MAY** attempt to log out from the OP using the OpenID Connect RP-Initiated Logout protocol [OIDCL] if that protocol is supported by the OP. In any case, to prevent abuse before the cookie times out, an RDAP server **SHOULD** invalidate the HTTP cookie associated with the session as part of terminating the session.

An example "farv1_session/logout" request:

https://example.com/rdap/farv1_session/logout

The response to this request **MUST** be a valid RDAP response per [RFC9083]. It **MUST NOT** include any members that relate to a specific RDAP object type (e.g., "events" or "status"). In addition, the response **MAY** include an indication of the requested operation's success or failure in the "notices" data structure. The "notices" data structure **MAY** include an indication of the success or failure of any attempt to logout from the OP or to revoke the tokens issued by the OP.

An example of a "farv1_session/logout" response:

```
{
  "rdapConformance": [
    "farv1"
  ],
  "lang": "en-US",
  "notices": [
    {
      "title": "Logout Result",
      "description": [
        "Logout succeeded"
        "Provider logout failed: Not supported by provider.",
        "Token revocation successful."
      ]
    }
  ]
}
```

Figure 16

In the absence of a "logout" request, an RDAP session **MUST** be terminated by the RDAP server after a server-defined period of time. The server **SHOULD** also take appropriate steps to ensure that the tokens associated with the terminated session cannot be reused. This **SHOULD** include revoking the tokens or logging out from the OP if either operation is supported by the OP.

5.6. Request Sequencing

The requests described in this document are typically performed in a specific sequence: "farv1_session/login" (or the related "farv1_session/device" and "farv1_session/devicepoll" requests) to start a session, "farv1_session/status" and/or "farv1_session/refresh" to manage a session, and "farv1_session/logout" to end a session. If a client sends a "farv1_session/status", "farv1_session/refresh", or "farv1_session/logout" request in the absence of a session cookie, the server **MUST** return an HTTP 409 (Conflict) error.

A client can end a session explicitly by sending a "farv1_session/logout" request to the RDAP server. A session can also be ended implicitly by the server after a server-defined period of time. The status of a session can be determined at any time by sending a "farv1_session/status" query to the RDAP server.

An RDAP server **MUST** maintain session state information for the duration of an active session. This is commonly done using HTTP cookies as described in [RFC6265]. Doing so allows End-Users to submit queries without having to explicitly identify and authenticate themselves for every query.

An RDAP server can receive queries that include a session cookie where the associated session has expired or is otherwise unavailable (e.g., due to the user requesting explicit logout for the associated session). The server **MUST** return an HTTP 401 (Unauthorized) error in response to such queries.

6. Protocol Features for Token-Oriented Clients

This specification adds additional processing steps for token-oriented clients as described in this section and Section 3.1.3. It does not define additional data structures or RDAP-specific protocol parameters specifically for token-oriented clients.

6.1. Client Login

Clients identify and authenticate End-Users by exchanging information with an OP that is recognized by the RDAP server as described in Sections 3.1.4.2, 3.1.4.3, and 3.1.4.4. A client **SHOULD** append the "additionalAuthorizationQueryParams" values retrieved from the "openidcProviders" array described in Section 4.1 to the Authorization Endpoint URL when requesting authorization from the OP. Once these processes are completed successfully, the client can request tokens from the OP as described in Section 3.1.4.5. The OP **SHOULD** include the RDAP server's client_id in the "aud" claim value of an issued ID Token. The RDAP server **MAY** choose to ignore the value of the "aud" claim or exchange the token as described in Section 6.4. With these steps completed, the Access Token received from the OP can be passed to an RDAP server in an HTTP "Authorization" request header [RFC6750] for RDAP queries that require End-User identification, authentication, and authorization.

6.2. Client Queries

An RDAP server that receives a bearer token in an HTTP "Authorization" request header as part of an RDAP object query **MUST** validate the token in accordance with local policy and confirm that the token is a legitimate Access Token. Once validated, the Access Token **MAY** be used to retrieve the claims associated with the End-User's identity, including claims associated with the "rdap" scope that are not already included in the Access Token, as described in Section 3.1.4.6. The RDAP server can then evaluate the End-User's identity information to determine the End-User's authorization level and process the query in accordance with server policies. A client **MUST** include the "farv1_iss" query parameter and issuer identifier value with an RDAP query if the token was issued by a remote OP.

6.3. Access Token Validation

An RDAP server **MUST** validate a received Access Token prior to using that token for access control purposes. Validation **MAY** include token introspection [RFC7662] using the issuing OP or analysis of the values included in a JWT Access Token. Once an Access Token is validated, an RDAP server **MAY** use that token to request user claims from the issuing OP.

There are performance considerations associated with the process of validating a token and requesting user claims as part of processing every received RDAP query. An RDAP server **MAY** cache validated information and use that cached information to reduce the amount of time needed to process subsequent RDAP queries associated with the same Access Token as long as the token has not expired. The client **SHOULD** monitor the token expiration time and refresh the token as needed.

6.4. Token Exchange

Tokens can include an "aud" (audience) claim that contains the OAuth 2.0 client_id of the RP as an audience value. In some operational scenarios (such as a client that is providing a proxy service), an RP can receive tokens with an "aud" claim value that does not include the RP's client_id. These tokens might not be trusted by the RP, and the RP might refuse to accept the tokens. This situation can be remedied by having the RP exchange the Access Token with the OP for a set of trusted tokens that reset the "aud" claim. The token exchange protocol is described in [RFC8693].

7. RDAP Query Processing

Once an RDAP session is active, an RDAP server **MUST** determine if the End-User is authorized to perform any queries that are received during the duration of the session. This **MAY** include rejecting queries outright, and it **MAY** include omitting or otherwise redacting information that the End-User is not authorized to receive. Specific processing requirements are beyond the scope of this document.

8. RDAP Conformance

RDAP responses that contain values described in this document **MUST** indicate conformance with this specification by including an rdapConformance [RFC9083] value of "farv1" (Federated Authentication for RDAP version 1). The information needed to register this value in the "RDAP Extensions" registry is described in Section 9.1.

Example rdapConformance structure with extension specified:

```
"rdapConformance" :
  [
    "rdap_level_0",
    "farv1"
  ]
```

Figure 17

9. IANA Considerations

9.1. RDAP Extensions Registry

IANA has registered the following value in the "RDAP Extensions" registry:

Extension Identifier: farv1
Registry Operator: Any

Specification: RFC 9560
Contact: IETF <iesg@ietf.org>
Intended Usage: This extension describes version 1 of a federated authentication method for RDAP using OAuth 2.0 and OpenID Connect.

9.2. JSON Web Token Claims Registry

IANA has registered the following values in the "JSON Web Token Claims" registry:

Claim Name: rdap_allowed_purposes
Claim Description: This claim describes the set of RDAP query purposes that are available to an identity that is presented for access to a protected RDAP resource.
Change Controller: IETF
Reference: Section 3.1.5.1 of RFC 9560.

Claim Name: rdap_dnt_allowed
Claim Description: This claim contains a JSON boolean literal that describes a "do not track" request for server-side tracking, logging, or recording of an identity that is presented for access to a protected RDAP resource.
Change Controller: IETF
Reference: Section 3.1.5.2 of RFC 9560.

9.3. RDAP Query Purpose Registry

IANA has created a new protocol registry to manage RDAP query purpose values.

Section at https://www.iana.org/protocols: Registration Data Access Protocol (RDAP)
Registry Name: Registration Data Access Protocol (RDAP) Query Purpose Values
Registration Procedure(s): This registry is operated under the "Specification Required" policy defined in [RFC8126]. The designated expert must ensure that requests to add values to this registry meet the syntax, value, and description requirements described in this section.
Required Information: Registration requests are described in a specification that's consistent with the "Specification Required" policy defined in [RFC8126]. The specification must include one or more purpose values as described below.

Individual purpose values are registered with IANA. Each entry in the registry contains the following fields:

Value: The purpose string value being registered. Value strings can contain uppercase ASCII characters from "A" to "Z", lowercase ASCII characters from "a" to "z", and the underscore ("_") character. Value strings contain at least one character and no more than 64 characters.
Description: One or two sentences in English describing the meaning of the purpose value, how it might be used, and/or how it should be interpreted by clients and servers.

The set of initial values used to populate the registry as described below are taken from the final report produced by the Expert Working Group on gTLD Directory Services chartered by the Internet Corporation for Assigned Names and Numbers (ICANN).

Value: domainNameControl
Description: Tasks within the scope of this purpose include creating and managing and monitoring a registrant's own domain name, including creating the domain name, updating information about the domain name, transferring the domain name, renewing the domain name, deleting the domain name, maintaining a domain name portfolio, and detecting fraudulent use of the registrant's own contact information.

Value: personalDataProtection
Description: Tasks within the scope of this purpose include identifying the accredited privacy/proxy provider associated with a domain name, reporting abuse, requesting reveal, or otherwise contacting the provider.

Value: technicalIssueResolution
Description: Tasks within the scope of this purpose include (but are not limited to) working to resolve technical issues, including email delivery issues, DNS resolution failures, and website functionality issues.

Value: domainNameCertification
Description: Tasks within the scope of this purpose include a Certification Authority (CA) issuing an X.509 certificate to a subject identified by a domain name.

Value: individualInternetUse
Description: Tasks within the scope of this purpose include identifying the organization using a domain name to instill consumer trust or contacting that organization to raise a customer complaint to them or file a complaint about them.

Value: businessDomainNamePurchaseOrSale
Description: Tasks within the scope of this purpose include making purchase queries about a domain name, acquiring a domain name from a registrant, and enabling due diligence research.

Value: academicPublicInterestDNSResearch
Description: Tasks within the scope of this purpose include academic public interest research studies about domain names published in the registration data service, including public information about the registrant and designated contacts, the domain name's history and status, and domain names registered by a given registrant (reverse query).

Value: legalActions

Description: Tasks within the scope of this purpose include investigating possible fraudulent use of a registrant's name or address by other domain names, investigating possible trademark infringement, contacting a registrant/licensee's legal representative prior to taking legal action, and then taking a legal action if the concern is not satisfactorily addressed.

Value: regulatoryAndContractEnforcement
Description: Tasks within the scope of this purpose include tax authority investigation of businesses with online presences, Uniform Domain-Name Dispute-Resolution Policy (UDRP) investigation, contractual compliance investigation, and registration data escrow audits.

Value: criminalInvestigationAndDNSAbuseMitigation
Description: Tasks within the scope of this purpose include reporting abuse to someone who can investigate and address that abuse or contacting entities associated with a domain name during an offline criminal investigation.

Value: dnsTransparency
Description: Tasks within the scope of this purpose involve querying the registration data made public by registrants to satisfy a wide variety of use cases around informing the public.

10. Security Considerations

Security considerations for RDAP can be found in [RFC7481]. Security considerations for OpenID Connect Core [OIDCC] and OAuth 2.0 [RFC6749] can be found in their reference specifications; best current security practice for OAuth 2.0 can be found in [OAUTH-SECURITY]. Additionally, the practices described in [RFC9325] **MUST** be followed when the Transport Layer Security (TLS) protocol is used.

As described in Section 3.1.4.2, the OAuth 2.0 Implicit Flow [RFC6749] is considered insecure, and efforts are being made to deprecate the flow. It **MUST NOT** be used.

Some of the responses described in this specification return information to a client from an RDAP server that is intended to help the client match responses to queries and manage sessions. Some of that information, such as the "userClaims" described in Section 5.1.1, can be personally identifiable and considered sensitive if disclosed to unauthorized parties. An RDAP server operator must develop policies for information disclosure to ensure that personally identifiable information is disclosed only to clients that are authorized to process that information.

The "do not track" claim relies on the good will of the RDAP server and associated proxies. As such, using and processing this claim depends on out-of-band trust relationships that need to be established before the claim is used in practice. If used and accepted by the RDAP server, there is a risk of information loss that could seriously impair audit capabilities.

10.1. Authentication and Access Control

Having completed the client identification, authorization, and validation process, an RDAP server can make access control decisions based on a comparison of client-provided information (such as the set of "userClaims" described in Section 5.1.1) and local policy. For example, a client who provides an email address (and nothing more) might be entitled to receive a subset of the information that would be available to a client who provides an email address, a full name, and a stated purpose. Development of these access control policies is beyond the scope of this document.

11. References

11.1. Normative References

[HTMLURL] WHATWG, "URL (Living Standard)", March 2024, <https://url.spec.whatwg.org/>.

[OIDCC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OIDCD] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[OIDCL] Jones, M., de Medeiros, B., Agarwal, N., Sakimura, N., and J. Bradley, "OpenID Connect RP-Initiated Logout 1.0", September 2022, <https://openid.net/specs/openid-connect-rpinitiated-1_0.html>.

[OIDCR] Sakimura, N., Bradley, J., and M. Jones, "OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-registration-1_0.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

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

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

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/info/rfc7009>.

[RFC7480] Newton, A., Ellacott, B., and N. Kong, "HTTP Usage in the Registration Data Access Protocol (RDAP)", STD 95, RFC 7480, DOI 10.17487/RFC7480, March 2015, <https://www.rfc-editor.org/info/rfc7480>.

[RFC7481] Hollenbeck, S. and N. Kong, "Security Services for the Registration Data Access Protocol (RDAP)", STD 95, RFC 7481, DOI 10.17487/RFC7481, March 2015, <https://www.rfc-editor.org/info/rfc7481>.

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

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <https://www.rfc-editor.org/info/rfc7617>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8628] Denniss, W., Bradley, J., Jones, M., and H. Tschofenig, "OAuth 2.0 Device Authorization Grant", RFC 8628, DOI 10.17487/RFC8628, August 2019, <https://www.rfc-editor.org/info/rfc8628>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/info/rfc8693>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <https://www.rfc-editor.org/info/rfc9068>.

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

[RFC9083] Hollenbeck, S. and A. Newton, "JSON Responses for the Registration Data Access Protocol (RDAP)", STD 95, RFC 9083, DOI 10.17487/RFC9083, June 2021, <https://www.rfc-editor.org/info/rfc9083>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9325] Sheffer, Y., Saint-Andre, P., and T. Fossati, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 9325, DOI 10.17487/RFC9325, November 2022, <https://www.rfc-editor.org/info/rfc9325>.

11.2. Informative References

[OAUTH-SECURITY] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "OAuth 2.0 Security Best Current Practice", Work in Progress, Internet-Draft, draft-ietf-oauth-security-topics-25, 8 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-25>.

[OIDC] OpenID, "What is OpenID Connect", <https://openid.net/developers/how-connect-works/>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/info/rfc8792>.

Acknowledgments

The author would like to acknowledge the following individuals for their contributions to the development of this document: Julien Bernard, Marc Blanchet, Tom Harrison, Russ Housley, Jasdip Singh, Rhys Smith, Jaromir Talir, Rick Wilhelm, and Alessandro Vesely. In addition, the Verisign Registry Services Lab development team of Joseph Harvey, Andrew Kaizer, Sai Mogali, Anurag Saxena, Swapneel Sheth, Nitin Singh, and Zhao Zhao provided critical "proof of concept" implementation experience that helped demonstrate the validity of the concepts described in this document.

Pawel Kowalik and Mario Loffredo provided significant text contributions that led to welcome improvements in several sections of this document. Their contributions are greatly appreciated.

Author's Address

Scott Hollenbeck
Verisign Labs
12061 Bluemont Way
Reston, VA 20190
United States of America
Email: shollenbeck@verisign.com
URI: https://www.verisignlabs.com/