Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 28, 2020

A. Malhotra
D. Schwartz
Ripple
May 27, 2020

PayID Protocol
draft-aanchal-payid-protocol-01

Abstract

This specification defines the PayID protocol - an application-layer protocol, which can be used to interact with a PayID-enabled service provider. The primary use-case is to discover payment account information along with optional metadata identified by a PayID [PAYID-URI]. The protocol is based on HTTP transfer of PayID protocol messages over a secure transport.

Feedback

This specification is a part of the PayID Protocol [1] work. Feedback related to this specification should be sent to payid@ripple.com [2].

Status of This Memo

Copyright Notice

Table of Contents

1. Terminology

This protocol can be referred to as "Basic PayID Protocol" or "PayID Protocol". The following terminology is used by this specification.

o Endpoint: either the client or the server of the connection.

  * Sending Endpoint: sending side of the transaction (wallet or exchange).

  * Receiving Endpoint: receiving side of the transaction (wallet or exchange).

o PayID client: The endpoint that initiates PayID protocol.

o PayID owner: The owner of the PayID URI as described in [PAYID-URI].

o PayID server: The endpoint that returns payment account information.

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 [RFC2119] and [RFC9174][].

2. Introduction

[PAYID-URI] describes a URI scheme to identify payment account(s) at a service provider. [PAYID-DISCOVERY], on the other hand, defines how to transform a PayID URI into a PayID URL that can be used by other protocols to interact with a PayID-enabled service provider but does not define protocol(s) to do so.

This document specifies PayID protocol - an application-layer protocol which can be used to interact with a PayID-enabled service provider identified by PayID URL using standard HTTP methods over a secure transport. In its most basic mode, a PayID protocol resource returns a JavaScript Object Notation (JSON) object representing the payment account(s) information along with optional metadata corresponding to the queried PayID URI [PAYID-URI]. The protocol defines new media formatting types for requests and responses but uses normal HTTP content negotiation mechanisms for selecting alternatives that the PayID client and server may prefer in anticipation of serving new use cases.

2.1. Design Goals

o Extensibility

Although, the primary use-case for payment account(s) information resource returned via Basic PayID protocol is assumed to be for making payments, the PayID protocol is designed to be easily extensible to facilitate creation and retrieval of other resources about the PayID owner, PayID client and/or PayID server that might be required for making payments.

o Neutrality: Currency and Network Agnostic

PayID protocol is designed to be a fundamentally neutral protocol. PayID protocol is capable of returning a PayID owner’s payment account(s) information for any network that they (or their service) support. This makes PayID protocol a network and currency agnostic protocol, capable of enabling payments in BTC, XRP, ERC-20 tokens, Lightning, ILP, or even fiat networks like ACH.

o Decentralized & Peer-to-Peer

Just like email servers, anyone can run their own PayID server or use third-party hosted services. If self-hosted, PayID protocol introduces no new counterparty risk or changes to a service’s security or privacy model. PayID protocol doesn’t require new, complex, and potentially unreliable peer discovery protocols, instead establishing direct peer-to-peer connections between communicating parties from the start. PayID is built on the most successful decentralized network: the web. There is no designated centralized authority, or a risk of a patchwork of different standards in different jurisdictions that make a global solution impossibly complex.

o Service Sovereignty

Each service provider that uses PayID for their users maintains full control of its PayID URL space, PayID service and has the ability to incorporate any policy they choose, including privacy, authentication, and security. They also have full sovereignty over users on their domain, just like in email. PayID is highly generalized and does not prescribe any particular solution outside of the standardized communication, which makes it compatible with existing compliance and user management tools and philosophies.

3. PayID Server Discovery

To support PayID protocol, the PayID client needs to discover the PayID URL corresponding to the PayID URI [PAYID-URI]. This can be obtained either using mechanisms described in [PAYID-DISCOVERY] or could be entered manually.

4. JSON Format Design

JSON as described in [RFC8259][], defines a test format for serializing structured data. Objects are serialized as an unordered collection of name/value pairs. JSON does not define any semantics around the name/value pairs that make up an object. PayID protocol’s JSON format defines name/value pairs that annotate a JSON object, property or array for PayID protocol resources.

The PayID client MUST request PayID response in JSON format through the "Accept" header with the media type as defined below, optionally followed by format parameters. One of the optional parameters is the case insensitive "q" value as described in Section 5.3.1 of [RFC7231].

Each message body is represented as a single JSON object. This object contains name/value pair whose value is the correct representation for a primitive value as described in [RFC8259][], or an array or object as described in section below.

If the PayID server does not support JSON format, it MUST reply with an appropriate error response.

4.1. HTTP Method

The PayID protocol payment account(s) information resource is requested using the HTTP GET method.

Following are the media types to request the payment account(s) information resource on different payment-networks and environments.

4.2. Media Type of Payment Account(s) Information Resource

The media type for payment account information resource is "application/* + json".

4.3. Response for application/* + json

The response body for "application/* + json" is a JSON object with the following name/value pairs.

```
  {
   optional string payId,
   required Address[] addresses,
   optional string memo,
   optional string identity,
   optional ProofOfControlSignature proofOfControlSignature
  }
```

4.3.1. payId

The value of "payId" field is the PayID URI in the client request that identifies the payment account information that the JSON object describes.

The "payId" field is an optional field in the response.

4.3.2. addresses

The value of "addresses" field is a JSON array of type "Address" of one or more JSON objects with the following name/value pairs.

```
  {
   required string paymentNetwork,
   optional string environment,
   required string addressDetailsType,
   required addressDetailsType addressDetails
  }
```

- paymentNetwork: The value of payment-network as specified in the client request’s "Accept" header

- environment: The value of environment as specified in the client request’s "Accept" header

- addressDetailsType: The value of "addressDetailsType" is JSON object of one of the following types.

  * CryptoAddressDetails

   * ACHAddressDetails

- addressDetails: The value of "addressDetails" is the address information necessary to send payment on a specific payment-netowrk and environment.

The "addresses" field MUST be present in the response.

4.3.2.1. addressDetailsType

We define the following two types of payment address types.

- CryptoAddressDetails: This is a JSON object with the following name/value pairs.

  { required string address, optional string tag }

  * address: The value of "address" field contains the on-ledger address corresponding.

  * tag: The value of "tag" field is the tag value used by some cryptocurrencies to distinguish accounts contained within a singular address. E.g XRP’s destination tag.

- ACHAddressDetails: This is a JSON object with the following name/value pairs.

  { required string accountNumber, required string routingNumber }

  * accountNumber: The value of "accountNumber" is the ACH account number.

  * routingNumber: The value of "routingNumber" is the ACH routing number.

4.3.3. memo

The "memo" string may specify additional metadata corresponding to a payment.

The "memo" string is an OPTIONAL field in the response.

4.3.4. identity

The "identity" string may specify any additional identity information about the PayID owner or PayID server.

The "identity" string is an OPTIONAL field in the response.

4.3.5. proofOfControlSignature

The value of "proofOfControlSignature" field is a JSON object of type "ProofOfControlSignature" with the following name/value pairs.

```
  {
   required string publicKey,
   required string payID,
   required string hashAlgorithm,
   required string signature
  }
```

- o publicKey: On-ledger public key of the Receiving Endpoint

- o payID: PayID URI of the PayID owner

- o hashAlgorithm: Hash algorithms used to hash the entire contents of the "ProofOfControlSignature" message.

- o signature: Digital signature over the hash of the entire contents of the "ProofOfControlSignature" message using the private key corresponding to the public key in "publicKey". This is a proof that the Receiving Endpoint is the owner of the on-ledger public key in "publicKey".

The "proofOfControlSignature" is an OPTIONAL field in the response.

4.4. Meaning of Media Type application/* + json

"*" may represent different payment-networks and environments. In this document, we propose standards with the media types specific to XRP, ILP, and ACH payment networks. We also propose media type that returns all addresses across all payment networks. Other payment networks MUST establish standard media types for their networks at IANA.

- o Accept: application/payid-json Returns all payment account(s) information correspnding to the requested PayID URI

- o Accept: application/xrpl-mainnet+json

Returns XRPL mainnet XAddresses

- o Accept: application/xrpl-testnet+json

Returns XRPL testnet XAddresses

- o Accept: application/xrpl-devnet+json

Returns XRPL devnet XAddresses

- Accept: application/interledger-testnet+json

Returns mainnet payment pointer to initiate SPSP request

- Accept: application/interledger-devnet+json

Returns testnet payment pointer to initiate SPSP request

- Accept: application/ach+json

Returns account and routing number

The PayID client MAY specify more than one media types along with the preference parameter. The server MUST respond as described in the Content Negotiation section below.

5. Header Fields

PayID protocol defines semantics around the following request and response headers. Additional headers MAY be defined, but have no unique semantics defined in the PayID protocol.

5.1. Common Headers

The following headers are common between PayID requests and responses.

5.1.1. Header Content-Type

PayID requests and responses with a JSON message body MUST have a "Content-Type" header value of "application-json".

5.1.2. Header Content-Length

As defined in [RFC7230][], a request or response SHOULD include a "Content-Length" header when the message’s length can be determined prior to being transferred. PayID protocol does not add any additional requirements over HTTP for writing Content-Length.

5.1.3. Header PayID-version (TODO)

The PayID client MUST include the PayID version request header field to specify the version of the PayID protocol used to generate the request.

If present on a request, the PayID server MUST interpret the request according to the rules defined in the specified version of the PayID protocol or fail the request with an appropriate error response code.

If not specified in a request, the PayID server MUST fail the request with an appropriate error code.

5.2. Request Headers

In addition to common Headers, the PayID client MUST specify the following request header.

5.2.1. Header Accept

The PayID client MUST specify the "Accept" request header field with at least one of the registered media types (Section X). The purpose of this header is to indicate what type of content can be understood in the response. It specifies the "payment-network" and "environment" of the payment account and its representation format for which the PayID client wants to receive information. The representation format is always JSON.

PayID server MUST reject formats that specify unknown or unsupported format parameters.

5.3. Response Headers

In addition to the Common Headers, the PayID server MUST specify the following response header.

5.3.1. Header Cache-Control

PayID server MUST include the "Cache-Control" header with the max-age limit of 0. The intermediate hops or PayID client MUST not cache the responses.

6. Basic PayID Protocol

Basic PayID protocol is used to request payment account(s) information resource identified by a PayID URI from a PayID-enabled service provider identified by a PayID URL using HTTP over secure transport. When successful, PayID protocol always returns the JSON representation of payment account(s) information resource along with optional metadata. This information can be used for any purposes outside the scope of this document.

Basic PayID protocol comprise of request and response messages, each of which is defined in more detail below. The following is a visual representation of the basic protocol flow:

```
PayID client                                         PayID server
   |                                                          |
   |              1.) GET request to PayID URL                |
   |--------------------------------------------------------->|
   |                                                          | |
                                               |
   |              2.) 200 Ok                                  |
   |                  Payment account information response    |
   |<---------------------------------------------------------|
   |                                                          |
   |                                                          |
```

6.1. Step 1: HTTP Request to PayID URL using HTTP GET Method

A basic PayID client issues a query using the HTTP GET method to the PayID URL without any query parameters and body.

The PayID client MUST query the PayID server using HTTPS only. [RFC2818] defines how HTTPS verifies the PayID server’s identity. If the HTTPS connection cannot be established for any reason, then the PayID client MUST accept that the PayID request has failed and MUST NOT attempt to reissue the PayID request using HTTP over a non-secure connection.

6.2. Step 2: Payment Account Information Response

In response, the PayID server returns a JSON object representation of the payment account(s) information resource for the payment-network and environment requested by PayID client in the request "Accept" header field along with other required and/or optional meta data.

PayID servers MUST be able to process "application/payid+json" header type.

If the PayID server does not contain the payment account information corresponding to the request, the PayID server MUST respond with an appropriate error message.

A PayID server MAY redirect the PayID client; if it does, the redirection MUST only be to an "https" URI and the PayID client MUST perform certificate validation again when redirected.

6.3. Step 3: Parse Payment Account Information Response

If the PayID server returns a valid response, the response will contain one or more of fields defined above.

7. Example Use of Basic PayID Protocol

This section shows sample use of Basic PayID protocol in several hypothetical scenarios.

7.1. Basic PayID Protocol by a Wallet

Suppose Alice wishes to send a friend some XRP from a web-based wallet provider that Alice has an account on. Alice would log-in to the wallet provider and enter Bob’s PayID (say, "bob$receiver.example.com") into the wallet UI to start the payment. The Wallet application would first discover the PayID URL for the PayID service-provider using one of the mechanisms described in PayID discovery [PAYID-DISCOVERY] protocol.

The Wallet application would then issue an HTTPS GET request:

```
 GET /users/bob HTTP/1.1
 Host: www.receiver.example.com
 Accept: application/payid+json
 PayID-version: 0.1
```

The PayID server might respond like this:

```
 HTTP/1.1 200 OK
 Content-Type: application/json
 Content-Length: 403
 PayID-version: 0.1
 Cache-Control: max-age=0
 Server: Apache/1.3.11
 {
   "payId" : "bob$receiver.example.com",
   "addresses" :
   [
     {
       "paymentNetwork" : "xrpl",
       "environment" : "testnet",
       "addressDetailsType" : "CryptoAddressDetails",
       "addressDetails" : {
                    "address" : "XTVQWr6BhgBLW2jbFyqqufgq8T9eN7KresB684ZSHKQ
3oDth"
             }
      },
     {
       "paymentNetwork" : "ach",
       "environment" : "mainnet",
       "addressDetailsType" : "ACHAddressDetails",
       "addressDetails" : {
                    "accountNumber" : "5674536253",
                    "routingNumber" : "aYJTDFGLKAJD"
             }
      }
    ],
    "memo" : "Additional optional Information"
 }
```

In the above example we see that the PayID server returned a list of payment accounts identified by PayID "bob$receiver.example.com". This is because Alice’s Wallet asked for all the payment accounts corresponding to the PayID in the "Accept" header. Alice’s Wallet MAY then use the payment account information to make payments.

Another example:

```
 GET /users/bob HTTP/1.1
 Host: www.receiver.example.com
 Accept: application/xrpl-testnet+json; q=0.4,
         application/ach+json; q=0.1
 PayID-version= 0.1
```

The PayID server might respond like this:

```
 HTTP/1.1 200 OK
 Content-Type: application/json
 Content-Length: 403
 PayID-version: 0.1
 Cache-Control: max-age=0
 Server: Apache/1.3.11
 {
   "payId" : "bob$receiver.example.com",
   "addresses" :
   [
     {
       "paymentNetwork" : "xrpl",
       "environment" : "testnet",
       "addressDetailsType" : "CryptoAddressDetails",
       "addressDetails" : {
                    "address" : "XTVQWr6BhgBLW2jbFyqqufgq8T9eN7KresB684ZSHKQ
3oDth"
            }
      }
    ]
 }
```

8. Common Response Status Codes (TODO)

A PayID server MAY respond to request using any valid HTTP response code appropriate for the request. The PayID server SHOULD be as specific as possible in its choice of an HTTP specific status code.

8.1. Success Responses

The following response codes represent successful requests.

8.1.1. Response Code 200 OK

A request that does not create a resource returns 200 OK if it is completed successfully and the value of the resource is not null. null. In this case, the response body MUST contain the value of the resource specified in the request URL.

8.1.2. Response Code 3xx Redirection

As per [RFC7231], a 3xx Redirection indicates that further action needs to be taken by the client in order to fulfill the request. In this case, the response SHOULD include a Location header, as appropriate, with the URL from which the result can be obtained; it MAY include a Retry-After header.

8.2. Client Error Responses

Error codes in the 4xx range indicate a client error, such as a malformed request. In the case that a response body is defined for the error code, the body of the error is as defined for the appropriate format.

9. Content Negotiation

The PayID client MAY choose to query for all possible payment addresses corresponding to a PayID URI

```
GET / HTTP/1.1
Accept: application/all+json
```

In this case, the PayID server MUST respond with all payment account(s) information associated with the queried PayID.

Alternatively, the PayID client MAY choose to query for a subset payment account(s) information in the order of preference.

```
GET / HTTP/1.1
Accept: application/xrpl-testnet+json; q=0.4,
        application/xrpl-mainnet+json; q= 0.1
```

In this case, the PayID server MUST respond with the payment account(s) information corresponding to one of the payment-network and environment mentioned in the "Accept" header in the order of client request preference. If none of those exist, PayID server MUST send an appropriate error response.

Alternatively, the PayID client MAY combine the above two approaches.

```
GET / HTTP/1.1
Accept: application/xrpl-testnet+json; q=0.4,
        application/xrpl-mainnet+json; q= 0.1,
        application/payid+json
```

In this case, the PayID server MUST respond with the payment account information corresponding to one of the payment-network and environment mentioned in the "Accept" header in the order of PayID client’s preference. If none of those exist, then the PayID server MUST respond with payment account(s) information corresponding to all payment accounts associated with the queried PayID URI.

10. Versioning(TODO)

Versioning enables clients and servers to evolve independently. PayID protocol defines semantics for protocol versioning.

PayID requests and responses are versioned according to the PayID-version header. PayID clients include the PayID-version header in order to specify the maximum acceptable response version. PayID servers respond with the maximum supported version that is less than or equal to the requested "major"

11. Security Considerations

This security considerations section only considers PayID clients and servers bound to implementations as defined in this document. Such implementations have the following characteristics:

- o PayID URIs are static and well-known to the PayID client; PayID server URLs can be static or discovered.

The following is considered out-of-scope:

- o Communication between the PayID owner and the wallet or exchange (which acts as PayID server) for PayID URI registration, etc.

- o Communication between the sender of the transaction and PayID client to transfer information such as PayID URI and other transaction details, etc.

- o PayID server URL discovery by PayID client. Implementations using PayID-Discovery [PAYID-DISCOVERY] MUST consider the security considerations in the corresponding document.

- o PayID server URL resolution by PayID client. Implementations using DNS, DNSSEC, DoH, DoT, etc. MUST consider the security considerations of the corresponding documents.

11.1. Network Attacks

Basic PayID protocol’s security model assumes the following network attackers:

- o Off-path attacker: An off-path attacker can be anywhere on the network. She can inject and spoof packets but can not observe, or tamper with the legitimate traffic between the PayID client and the server.

- On-path attacker: An on-path attacker can eavesdrop, inject, spoof and replay packets, but can not drop, delay or tamper with the legitimate traffic.

- In-path or Man-in-the-middle (MiTM) attacker: An MiTM is the most powerful network attacker. An MiTM has full access to the communication path between the PayID client and the server. She can observe, modify, delay and drop network packets.

Additionally we assume that the attacker has enough resources to mount an attack but can not break the security guarantees provided by the cryptographic primitives of the underlying secure transport.

The basic PayID protocol runs over HTTPS and thus relies on the security of the underlying transport. Implementations utilizing TLS 1.3 benefit from the TLS security profile defined in [RFC 8446][] against all the above network attackers.

11.1.1. Denial-of-Service (DoS) attacks

As such cryptography can not defend against DoS attacks because any attacker can stop/interrupt the PayID protocol by: * Dropping network packets, * Exhaustion of resources either at the network level or at PayID client and/or server.

The PayID servers are recommended to follow general best network configuration practices to defend against such attacks [RFC4732].

11.2. Information Integrity

The HTTPS connection provides transport security for the interaction between PayID client and server but does not provide the response integrity of the data provided by PayID server. A PayID client has no way of knowing if data provided in the payment account information resource has been manipulated at the PayID server, either due to malicious behaviour on the part of PayID server administrator or as a result of being compromised by an attacker. As with any information service available on the Internet, PayID clients should be wary of the information received from untrusted sources.

12. Privacy Considerations

The PayID client and server should be aware that placing information on the Internet means that any one can access that information. While PayID protocol is an extremely useful tool to discovering payment account(s) information corresponding to a human-rememberable PayID URI, PayID owners should also understand the associated privacy risks. The easy access to payment account information via PayID protocol was a design goal of the protocol, not a limitation.

12.1. Access Control

PayID protocol MUST not be used to provide payment account(s) information corresponding to a PayID URI unless providing that data via PayID protocol by the relevant PayID server was explicitly authorized by the PayID owner. If PayID owner wishes to limit access to information, PayID servers MAY provide an interface by which PayID owners can select which information is exposed through the PayID server interface. For example, PayID servers MAY allow PayID owners to mark certain data as "public" and then utilize that marking as a means of determining what information to expose via PayID protocol. The PayID servers MAY also allow PayID owners to provide a whitelist of users who are authorized to access the specific information. In such a case, the PayID server MUST authenticate the PayID client.

12.2. Payment Address Rotation

The power of PayID protocol comes from providing a single place where others can find payment account(s) information correspondong to a PayID URI, but PayID owners should be aware of how easily payment account information that one might publish can be used in unintended ways. As one example, one might query a PayID server only to see if a given PayID URI is valid and if so, get the list of associated payment account information. If the PayID server uses the same payment address each time, it becomes easy for third-party to track one’s entire payment history. The PayID server SHOULD follow the best practice of payment address rotation for every query to mitigate this privacy concern.

12.3. On the Wire

PayID protocol over HTTPS encrypts the traffic and requires mutual authentication of the PayID client and the PayID server. This mitigates both passive surveillance [RFC7258] and the active attacks that attempt to divert PayID protocol queries to rogue servers.

Additionally, the use of the HTTPS default port 443 and the ability to mix PayID protocol traffic with other HTTPS traffic on the same connection can deter unprivileged on-path devices from interfering with PayID operations and make PayID traffic analysis more difficult.

12.4. In the PayID Server

The Basic PayID protocol data contains no information about the PayID client; however, various transports of PayID queries and responses do provide data that can be used to correlate requests. A Basic PayID protocol implementation is built on IP, TCP, TLS and HTTP. Each layer contains one or more common features that can be used to correlate queries to the same identity.

At the IP level, the PayID client address provides obvious correlation information. This can be mitigated by usee of NAT, proxy, VPN, or simple address rotation over time. It may be aggravated by use of a PayID server that can correlate real-time addressing information with other identifiers, such as when PayID server and other services are operated by the same entity.

PayID client implementations that use one TCP connection for multiple PayID requests directly group those requests. Long-lived connections have better performance behaviours than short-lived connections; however they group more requests, which can expose more information to correlation and consolidation. TCP-based solutions may also seek performance through the use of TCP Fast Open [RFC7413][]. The cookies used in TCP Fast open may allow PayID servers to correlate TLS connections together.

TCP-based implementations often achieve better handshake performance throught the use of some form of session resumption mechanism, such as Section 2.2 of [RFC8446]. Session resumption creates trivial mechanism for a server to correlate TLS connections together.

HTTP’s feature set can also be used for identification and tracking in a number of ways. For example, Authentication request header fields explicitly identify profiles in use, and HTTP cookies are designed as an explicit state-tracking mechanism and are often used as an authentication mechanism.

Additionally, the "User-Agent" and "Accept-Language" request header fields often convey specific information about the PayID client version or locale. This allows for content-negotiation and operational work-arounds for implementation bugs. Request header fields that control caching can expose state information about a subset of the client’s history. Mixing PayID queries with other HTTP requests on the same connection also provides an opportunity for richer data correlation.

The PayID protocol design allows implementations to fully leverage the HTTP ecosystem, including features that are not enumerated in this document. Utilizing the full set of HTTP featurees enables

PayID to be more than HTTP tunnel, but it is at the cost of opening up implementations to the full set of privacy considerations of HTTP.

Implementations of PayID clients and servers need to consider the benefits and privacy impacts of these fetaures, and their deployment context, when deciding whether or not to enable them. Implementations are advised to expose the minimal set of data needed to achieve the desired feature set.

Determining whether or not PayID client implementation requires HTTP cookie [RFC6265][] support is particularly important because HTTP cookies are the primary state tracking mechanism in HTTP, HTTP cookies SHOULD NOT be accepted by PayID clients unless they are explicitly required by a use case.

Overall, the PayID protocol does not introduce privacy concerns beyond those associated with using the underlying IP, TCP, TLS and HTTP layers.

13. IANA Considerations

This document defines registries for PayID protocol version and application/* +json media types.

13.1. Header Field Registration

Header field name: PayID-version: major.minor

Applicable Protocol: "PayID protocol"

Status: "standard"

Author/Change controller: Refer to the contact details of the authors in this document.

Specification Document: "PayID protocol"

13.2. Media Type Registration

This document registers multiple media types, listed in Table 1.

Type	Subtype	Specification
application	xrpl-mainnet+json	
application	xrpl-testnet+json	
application	xrpl-devnet+json	
application	ach+json	
application	interledger-mainnet+json	
application	interledger-testnet+json	
application	payid+json	

Type Name: application

Subtype name: This document registers multiple subtypes as listed in table 1

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See[RFC7159][].

Security considerations: Security considerations relating to the generation and consumption of PayID protocol messages are discussed in Section X.

Interoperability considerations: This document specifies format of conforming messages and the interpretation thereof.

Published specification: This document is the specification for these media types; see Table 1 for the section documenting each media type.

Applications that use this media type: PayID servers and PayID clients.

Additional information:

Magic number(s): n/a

File extension(s): This document uses the mime type to refer to protocol messages and thus does not require a file extension.

Macintosh file type code(s): n/a

Person & email address to contact for further information: See Authors’ Addresses section.

Intended usage: COMMON

Restrictions on usage: n/a

Author: See Authors’ Addresses section.

Change controller: Internet Engineering Task Force (mailto:iesg@ietf.org).

14. Acknowledgments

15. References

15.1. Normative References

[PAYID-DISCOVERY]
Fuelling, D., "PayID Discovery", n.d..

[PAYID-URI]
Fuelling, D., "The ’payid’ URI Scheme", n.d., <https://tbd.example.com/>.

[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>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

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

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

15.2. Informative References

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <https://www.rfc-editor.org/info/rfc4732>.

15.3. URIs

[1] https://payid.org/

[2] mailto:payid@ripple.com

Authors' Addresses

Aanchal Malhotra
Ripple
315 Montgomery Street
San Francisco, CA 94104
US

Phone: -----------------
Email: amalhotra@ripple.com
URI: https://www.ripple.com

David Schwartz
Ripple
315 Montgomery Street
San Francisco, CA 94104
US

Phone: -----------------
Email: david@ripple.com
URI: https://www.ripple.com