Authentication & Authorization Privacy Protecting Protocol

A2P3 Protocol Specification

Author: Dick Hardt, dick.hardt@gmail.com
Draft: 10, October 6, 2013

Abstract

This document describes the high level protocol for a User to authorize an App to interact with one or more Resource Servers on behalf of the User while protecting the User's privacy.

The flow starts with the App directing the User to the Agent with a Request. The Agent authenticates the User and prompts the User to authorize the App to interact with Resource Servers. The Agent then asks the Authentication Service to generate an authentication token, which the Agent then returns to the App. The App presents the authentication token and Request to the Identifier Exchange and receives an authorization token for each Resource Server. The App then presents an authorization token to each Resource Server to perform User authorized transactions.

Status

A2P3 is an experimental protocol for exploring the privacy, security, scalability and user experience issues in solving identity related use cases at internet scale.

Table of Contents

1. Protocol Objectives

1.1. Privacy Protecting

User data and identifiers are compartmentalized to minimize correlating User activity across services.

1.2. Open Platform

Any User can develop an App or Resource Server.

1.3. Simple Development

Implementation complexity is minimized for App and Resource Server developers.

1.4. Web and Mobile

An App can be a native mobile application or a web application.

1.5. Interoperable

There may be multiple deployments of each component, and they will easily interoperate. System components are loosely coupled and replaceable.

1.6. Credential Choice

The User can authenticate with a NFC enabled card or a mobile device. The User can easily enroll her mobile devices to be additional authentication credentials.

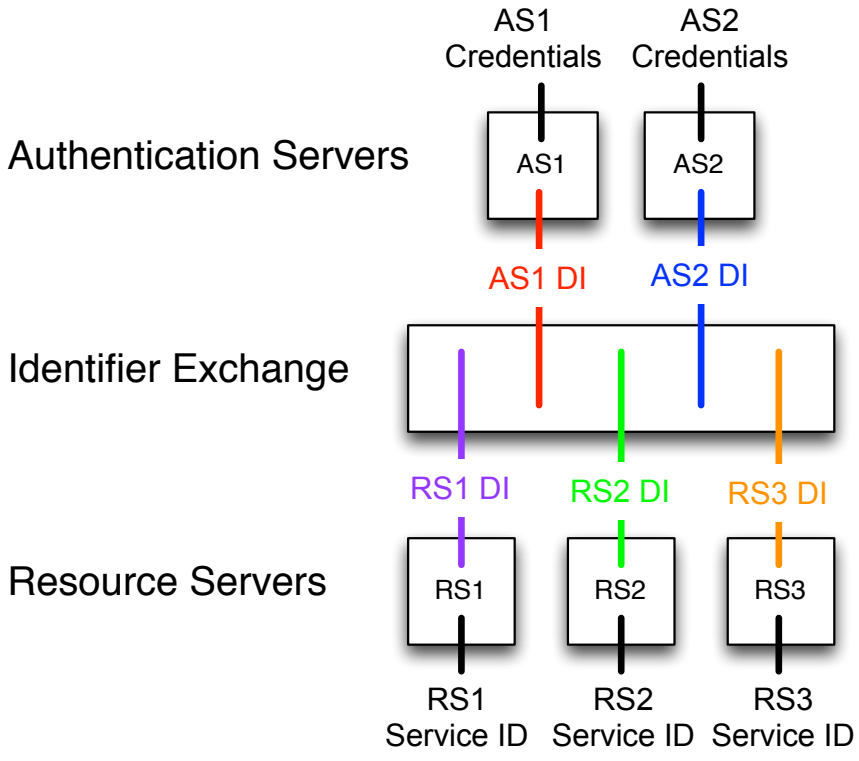
2. Privacy Model

A fundamental protocol requirement is to maintain the User's privacy. One aspect of this is to ensure the User making informed consent what PII the User is releasing and to which party. Another aspect is to ensure the system does not provide any of the parties with common identifiers that can be used to correlate User activity at multiple parties. Rather than having a common identifier for the User across the system, each party has a separate, Directed Identifier (DI) that it shares only with the Identifier Exchange.

The Authentication Servers have identifiers used to authenticate the User. After authenticating the User, the Authentication Server creates a token containing the DI it shares with the Identifier Exchange. The Identifier Exchange will then create a token for each requested Resource Server, with each corresponding token containing the DI shared between the Identifier Exchange and that specific Resource Server.

Colluding Authentication Servers or Resource Servers are not able to correlate Users with the identifiers they have from the Identifier Exchange, maintaining existing compartmentalization of information about the User. Figure 1 illustrates the identifier relationships. AS1 DI, AS2 DI, RS1 DI, RS2 DI and RS3 DI are the Directed Identifiers.

Figure 1: Identifier Privacy Management



If the User chooses to share a public identifier such as an email address, phone number, social insurance number, health number or aggregated PII such as name, date of birth and postal address, than any parties that have received those public identifiers can collude and correlate User activity. This is a common practice today in various industries and jurisdictions. By providing a means for the user to share only the attributes that are required for a transaction, the need for a relying party to acquire public identifiers and PII is diminished and User privacy is enhanced.

Additional explanation on which party sees which information is illustrated in Appendix A – Transaction and Identifier Containment Matrix.

3. Roles

3.1. User

- The User is the person interacting with the system that is explicitly or implicitly authenticating and providing authorization.

3.2. Operator

- In some use cases, the App is being run by an Operator who is doing a transaction where the User is the subject of the transaction.

3.3. Developer

- The Developer is a User with administrative rights to register and manage an App or Resource Server at the Registrar and Resource Servers.

3.4. App

- The App is the program the User is interacting with. The App is requesting the system to authenticate the User and provide authorization to access resources on behalf of the User. The App will have both a client and server component. The client and server can securely communicate with each other.
- The client component runs on the device the User is interacting with and makes requests of the Agent. The client may be as thin as a web browser or as sophisticated as a native application.
- The server component maintains the App's secrets, generates requests and interacts with the Identifier Exchange and Resource Servers.
- Each App has a system unique identifier in the form of a host name that is registered at the Registrar.

3.5. Agent

- The Agent acquires authentication credentials from the User and presents authorization requests for the User to accept or decline.
- The Agent is tightly coupled to one Authentication Server and can securely communicate with the Authentication Server.
- The Agent runs on the device the User is interacting with and is trusted by the User.
- General Agents are running on a device that does not belong to the User such as a kiosk or at a counter.
- Personal Agents run on a device the User controls such as a mobile device and are able to be part of the User's authentication credentials and manage previous authorizations.

3.6. Authentication Server

- The Authentication Server (AS) transforms credentials from the User into an IX Token containing a Directed Identifier it shares with the Identifier Exchange.
- The AS may provide IX Tokens to more than one Identifier Exchange.

- The Agent and AS are tightly coupled, but are separate components to protect the User's privacy. The AS only knows the User is authenticating. The AS does not see the App or Resource Server identifiers.
- The AS will enroll new Personal Agents into the system for the User.
- The AS has a globally unique identifier in the form of a hostname.

3.7. Identifier Exchange

- The Identifier Exchange (IX) maintains a table of correlated Directed Identifiers for each User. The IX only sees Directed Identifiers about the User, and cannot correlate those Directed Identifiers to personally identifiable information (PII) about the User. The IX exchanges an IX Token containing a Directed Identifier from the Authentication Server into an RS Token containing the correlated Directed Identifier for the Resource Server.
- The IX trusts one or more Authentication Servers and shares cryptographic keys with them.
- The IX maintains a list of Personal Agents enrolled for a User.
- The IX has a globally unique identifier in the form of a hostname, the IX ID.
- The IX has a list of Authentication Servers it trusts.

3.8. Resource Server

- The Resource Server (RS) is an App that provides one or more resources for a User. A resource may provide data about the User or perform operations for the User as requested by an App via the RS API.
- Any App wanting to access an RS must be registered with the RS.
- The RS manages and enforces the policy of which Apps can access which resources for which Users.
- The RS trusts the Identifier Exchange, and implicitly, the Authentication Server.
- Each RS has a system unique identifier in the form of a host name that is registered at the Registrar.
- The RS hosts a Resource Description document for each authorization scope it offers, where the host name of the URL used to retrieve the document is the system unique identifier for the RS.
- Some Resource Servers will offer identical resources for Users. A Standardized Resource describes these identical resources and publishes them at a system unique host name registered at the Registrar. See section 12 for more detail on Standardized Resources.

3.9. Registrar

- The Registrar manages the registration and key management for Apps, Resource Servers and Standardized Resources.
- The Registrar has a globally unique identifier in the form of a hostname.
- The Registrar and the Identifier Exchange are tightly coupled. They are two separate components to enhance the privacy and the security of the system.

4. Use Cases and User Experience

4.1. Personal Agent Enrollment

The User has a supported mobile device and would like to use it to authorize transactions. The User has a NFC enabled card and access to a machine with an NFC reader running the Personal Agent Enrollment App for the specific Personal Agent running on the User's mobile device.

(A) User runs Personal Agent Enrollment App.
(B) User taps NFC enabled card and enters the corresponding Passcode.
(C) Personal Agent Enrollment App presents interface to delete any previously enrolled mobile devices, and option to enroll a new mobile device.
(D) User selects enrolling a new mobile device.
(E) User is prompted for Passcode to use with the new Personal Agent.
(F) User enters new Passcode (may be same as other Passcodes).
(G) Personal Agent Enrollment App generates a QR code.
(H) User launches the Personal Agent on their mobile device and scans QR Code.
(I) User enters new Passcode into Personal Agent.
(J) Personal Agent confirms enrollment.

4.2. Counter Application

This is an App at the counter of a government office, a health care facility or a bank that needs to identify the User to conduct a transaction. The Operator is sitting behind the counter.

(A) User approaches counter.
(B) Purpose of App is communicated to the User either orally or in writing.
(C) User taps NFC enabled card on reader or scans QR code with mobile device.
(D) Operator confirms photo from Resource Server matches User.
(E) Operator proceeds with processing.

4.3. Kiosk

This is an App running on a dedicated device providing the User with an automated service.

(A) User picks language and initiates transaction.
(B) User taps NFC enabled card or scans QR code with mobile device.
(C) User enters Passcode.
(D) User authorizes transaction.
(E) Kiosk completes transaction.

4.4. Federated Physical Access Management

This is an App that is controlling physical access to a building, room or storage area. There may be multiple organizations that can authorize Users to have access. A Resource Server represents each organization.

(A) User taps NFC enabled card on reader or scans QR code with mobile device.
(B) App gets Tokens for all federated Resource Servers.
(C) App confirms a Resource Server has granted access to the User.
(D) The App allows entry to the User.

4.5. Mobile Identity Verification

This is an App running on the Operator's NFC capable mobile device. The Operator requires identity data about the User to perform a transaction.

(A) Operator communicates purpose of App to the User and requests User to authorize.
(B) User taps NFC enabled card on Operator's mobile device or scans QR code on Operator's mobile device.
(C) Operator confirms photo from Resource Server matches User.
(D) Operator proceeds with processing.

4.6. Mobile Application

This is an App the User runs on their mobile device. The App may be a native app or a web app. The User has also installed and enrolled an Agent on their mobile device. The App requires authorization from the User to conduct a transaction.

(A) User initiates transaction with App.
(B) App invokes Agent.
(C) Agent prompts User to enter Passcode if requested by App.
(D) Agent presents User with descriptions from each Resource Server about requested transactions and prompts User to authorize.
(E) User authorizes transaction.
(F) Agent invokes the App.
(G) App proceeds with functionality.

4.7. One Time PC Web Login with Mobile

The User wants to log into a web App running on a PC with no NFC reader using the Personal Agent the User has installed and enrolled on their mobile device. The App requires authorization from the User to conduct a transaction. The User has not used the App recently and does not expect to use the App again soon.

(A) User navigates to the web App in the browser on the PC and initiates a transaction with App.
(B) App displays a QR code representing a Request URL (the Request URL may also be displayed) and instructs User to scan the QR code.
(C) User opens their Personal Agent and scans the QR code (or types the Request URL into an input field).
(D) Personal Agent prompts User to enter Passcode if requested by App.
(E) Personal Agent presents User with descriptions from each Resource Server about the requested transactions and prompts User to authorize.
(F) User authorizes transaction.
(G) Agent redirects to browser on the mobile device with the Return URL.
(H) The App displays a message in the browser on the mobile device that it is completing the login process.
(I) The App displays on the PC that the login was successful. The browser window on the mobile device is closed. The App proceeds with functionality.

4.8. Frequent PC Web Login with Mobile

The User wants to log into a web App running on a PC with no NFC enabled card reader using the Personal Agent the User has installed and enrolled on their mobile device. The App requires authorization from the User to conduct a transaction. The User expects to use the App again in the near future and would like the browser on the PC to remember her so that subsequent logins are quicker.

The initial login is the same experience as 2.6 with the addition that the User has selected an option during login on the PC to remember her.
Subsequent logins proceed as follows:

(A) User navigates to the web App in the browser on the PC and initiates an identity transaction with App.

(B.1) The App displays a User recognizable identifier such as email or display name to indicate that she has logged on before to this App. If more than one User has been remembered, all User identifiers are also presented to the User.

(B.2) The User selects her identifier to initiate login. If her identifier is not presented, then she selects an option that takes her to 3.7(B)

(B.3) The App informs the User that her mobile device is being contacted.

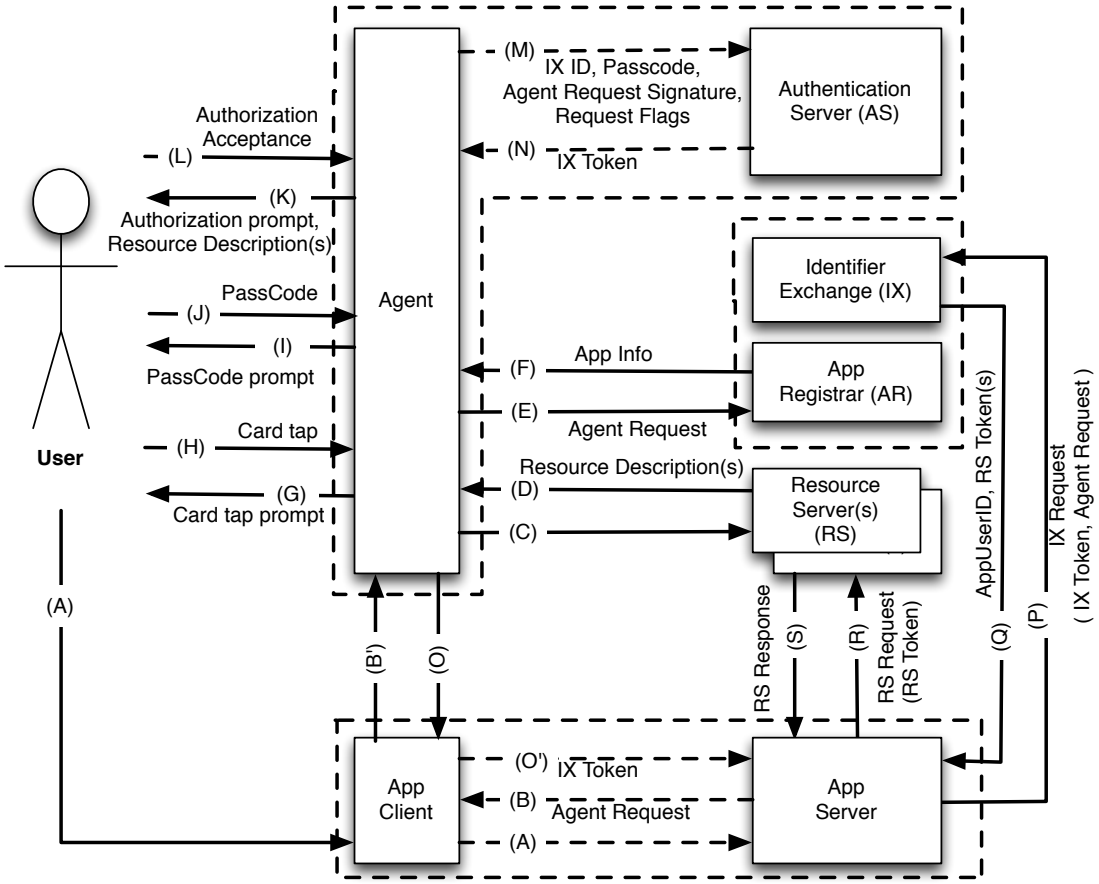
(C) The User's Personal Agent alerts her that an App would like to authenticate.

(D-I) are the same as 3.7(D-I)

5. App Flow

The flow illustrated in Figure 2 indicates how an App acquires User authorization to transact with a Resource Server. Components that are surrounded by dashed lines are tightly coupled and how they communicate is implementation dependant.

Figure 2: Typical App Authorization Flow



The specific steps are:

(A) The User initiates authentication and/or authorization with the App.

(B) The App Server creates and signs an Agent Request (details in Section 9. Requests) and sends it to the App Client. The App Client appends the Agent Request to the Agent API URL to create the Request URL and then invokes that URL. The Agent API URL has a custom scheme that is unique for the IX and SHOULD be the IX ID. The App may also encode the Request URL in a QR code so that the User can scan the QR code with their Personal Agent.
The Agent Request contains the App ID, the Resource Authorization (RA) URLs, a parameter indicating if the User is required to authorize the transaction, a parameter indicating how the User must authenticate and the App's Return URL.

(C) If the App provided any RA URLs and authorization is requested, the Agent fetches the RA URLs. This may be done in parallel with steps (E) – (J).

(D) The Resource Server returns the Resource Description. The Agent may cache the Resource Description as per HTTP caching headers.

(E) The Agent MAY send the Agent Request to the App Registrar for verification. This may be done in parallel with steps (C)-(D) and (G)-(J).
(F) The App Registrar:
    1) verifies the App is in good standing
    2) verifies the Agent Request has not expired
    3) verifies the Agent Request signature is valid
    4) returns the verification results along with the registered name and description of the App to the Agent.

    The Agent MAY cache the App ID and return URL to check future requests.
(G) If the Agent supports reading NFC enabled cards, the Agent prompts the User to tap their NFC enabled card.
(H) The User taps their NFC enabled card
(I) If the App requested a Passcode, the Agent prompts the User to enter their Passcode.
(J) The User enters their Passcode.
(K) If the App requested authorization, the Agent presents the User with the Resource Descriptions from any Resource Servers and prompts the User for authorization.
(L) The User agrees to the authorization.
(M) The Agent calls the Authentication Service with the IX ID (determined from the Agent URL custom scheme), the NFC enabled card or other Agent key material, and the signature from the Agent Request. The Agent The Agent MUST NOT send any other details of the transaction such as the App ID or the Resource Server IDs to the Authentication Service.
(N) The Authentication Service generates an IX Token (details in Section 10. Tokens) that includes the IX ID, the AS User ID, and the Agent Request signature and is encrypted with the Identifier Exchange's key and returns the IX Token to the Agent. See the Token Section for more detail on Token encodings.
(O) The Agent returns the IX Token and Agent Request to the App by redirecting to the Return URL.
(P) The App creates an IX Request with the Agent Request and the IX Token and presents it to the Identifier Exchange.
(Q) The Identifier Exchange:
    1) securely passes the IX Request to the Registrar (this flow not shown) which:
        i) verifies the IX Request.
        ii) verifies the Agent Request as in (F.1-4).
        iii) ensures the Agent Request does not contain any restricted RS IDs the App is not allowed to see.
        iv) returns encryption keys for all RSes in the Agent Request
    2) verifies the IX Token is valid, from an authorized Authentication Service and contains a known AS User ID and the Agent Request signature in the IX Token matches the signature of the passed in Agent Request.
    3) looks up the matching RS User ID for each Resource Server for the AS User ID.
    4) Generates a RS Token for each Resource Server including the authorization and authentication meta data from the IX Token.

5) returns the RS Tokens and App's User ID to the App

(R) For each RS Token, the App creates a RS Request and calls each Resource Server.

(S) The Resource Server:

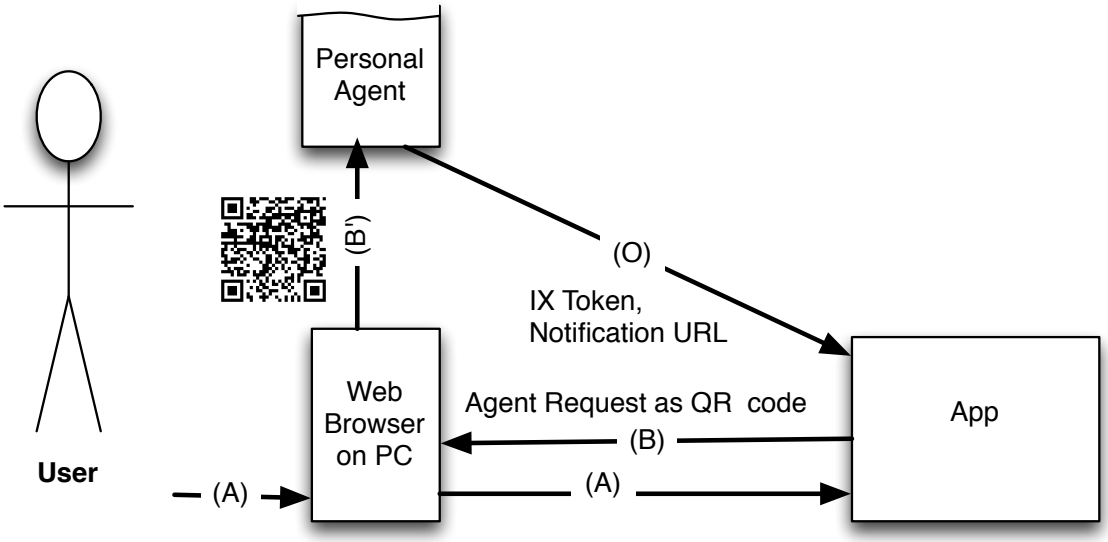
1) verifies the RS Request is valid
2) verifies the RS Token is valid
3) confirms the App ID in the RS Token matches the App ID in the RS Request.
4) evaluates the Resource Server's policy for the App, the RS Token authentication and authorization metadata, and if appropriate executes the transaction

NOTE: There may be additional implementation dependent calls between the Agent and the Authentication Service for setup and the timing of steps (K) and (L) may be earlier than (G)-(J) but must be after(C)-(D).

6. PC Web App Flow

Some Apps are best deployed as web applications run on a PC. As NFC readers are not widely available for PCs, the User can authorize a web application running on a PC by scanning a QR code with their Personal Agent running on their mobile device.

Figure 3: PC Web App Initial Flow



PC Web App Flow

Figure 3 shows the PC Web Initial Login Flow that satisfies use case 4.7. This flow works in the following manner:

(A) The User navigates to the URL for the App on a PC's browser. The User initiates logging into the App and the App generates a QR code representation of the Request URL. The App Server may optionally generate a Short URL that returns the Request URL as an HTTP 302 redirect to decrease the URL size for hand entry and QR Code encoding. PC Browser Session information is encoded as the state parameter.

(B) The User opens the Personal Agent on their mobile device and scans the QR Code, or types the Request URL into their Personal Agent. The Personal Agent converts the QR Code to a URL, parses the URL resolving the Short URL if needed, and confirms the URL scheme is correct. If the Personal Agent appends `?json=true` to the Short URL, the App returns the request as a JSON object.

(C-N) Are the same as in Figure 2

(O) The Agent redirects to the Return URL providing the IX Token and the Agent Request.

(P-S) Are the same as in Figure 2

7. Enrollment, Linking and Registration

7.1. Initial User Enrollment

Users are added to the system with a privileged App, an Enrollment App. When Users are initially added to the system, they are provided a credential from an Authentication Server and it is linked to existing accounts at one or more Resource Servers. One or more existing Public Identifiers must be correlated to the User Credentials being distributed. This correlation is captured by the Identifier Exchange as Directed Identifiers, and no entity knows the correlated account and credential identifiers after the User is enrolled.

(A) User engages in process where Enrollment App has confidence the User is correlated to one or more account identifiers from Resource Servers. This may occur in person at a counter where the User is providing various identity documents, or other situations where the User is able to assert their identity.

(B) Enrollment App acquires a credential identifier for the credentials from the Authentication Server.

(C) Enrollment App requests a correlated AS DI and RS DI for each Resource Server from the Identifier Exchange using `/di/create` per 10.5.

(D) Enrollment App sends credential identifier and AS DI to the Authentication Service and the matching RS DI and account identifiers to each Resource Server using `/di/link` per 10.6.

(E) Enrollment application expunges all knowledge of transaction.

7.2. User Account Linking

The User will have many pre-existing accounts at existing Apps. In order to the use the system at an existing App, the existing account needs to be linked to the App's Directed Identifier. The User will need to authenticate to the App using an App specific mechanism to enable the App to have high assurance which account is associated with the User. The account linking may rely on identity data about the User from other Resource Servers.

7.3. Personal Agent Enrollment

Personal Agents are able to be the User's credentials in the system. The user experience for registering a Personal Agent is described in 4.1.

The Personal Agent Enrollment App will obtain authorization from the User for the IX Personal Agent Enrollment URL, resulting in a RS Token for the IX. The Personal Agent Enrollment App is operated by the Authentication Server and calls APIs at the IX as described in 10.5.

7.4. App Registration

Each App has a unique identifier in the form of a domain name. In order to participate in the system, the Apps must register at the Identifier Exchange and then at any Resource Server's they wish interact with on behalf of the User.

7.4.1. App Registration at Registrar

(A) The Developer navigates to the Registrar and logs in the same as a regular App.
(B) The Registrar presents a dashboard to the Developer with an option to register a new App which the Developer selects.
(C) The Developer enters the new App's host name (App ID), return URL(s), name, description, Anytime Resource flag and agrees to the Identifier Exchange's Terms of Service.
(D) The Identifier Exchange checks that there is no other App with that ID and then may require the Developer to claim she has control over that host name.
(E) The Identifier Exchange then completes registration of the App and presents the Developer with the App's signing key and key id for creating Agent and IX Requests.
(F) The Developer can return at anytime to the Registrar and generate a new signing key and key id.

7.4.2. App Registration at Resource Server

(A) The Developer navigates to the Resource Server's registration page and logs in the same as a regular App authorization the Resource Server to access information at the Registrar.
(B) The Developer enters the App ID they would like to register at the Resource Server.
(C) The Resource Server calls `/app/verify` at the Registrar per section 10.3.
(D) If the Registrar confirms the Developer is authoritative for the App ID, the Resource Server prompts the Developer to accept its Terms of Service and then presents the Developer with the App's signing key and key id for creating RS Requests.
(E) There may be additional interactions between the Resource Server and the Developer to grant additional access to the App.
(F) The Developer can return at anytime to the Resource Server and generate a new signing key and key id.

7.4.3. Resource Server Registration

A Resource Server is registered at the Registrar the same as an App. RS Tokens generated by the Identifier Exchange will be encrypted with the same key as was provided in 7.4.2(F).

7.4.4. Administrator Management

At the Registrar the Developer may administer which other Developers have administrative access to the App. The Registrar presents some mechanism for the Developer to uniquely identify other Developers to add them as administrators. The Registrar will also enable a Developer to remove other Developers from having administrative access to an App.

8. Requests

Requests are a signed JSON token containing parameters used in making an API request. Requests are encoded per JWT[1] and signed per JWS[2] using the signing key shared between the two parties. Request tokens created by an App are defined in this section. Other APIs in the system use the same Request format. See the API documentation for the properties of the "request.a2.p3.org" object. All requests have the same header.

Example Header:

```
{ "typ":"JWS"
, "alg":"HS256"
, "kid":"yWe4gV2"
}
```

The payload must include:

"iat": date/time stamp when Request was generated as IntDate
"iss": App ID
"aud": ID of audience, either the Resource Server ID or the Identifier Exchange ID
"request.a2p3.org": object containing request parameters, and one of those must be:

8.1. Agent Request

The "aud" property will be the Identifier Exchange ID and the signing key and key id will be the one the App obtained from the Registrar. The "request.a2p3.org" object MUST contain the following properties:

"resources": an array of zero or more resource authorization URLs
"auth": an object containing authorization and authentication requests

The "request.a2p3.org" object MUST contain one of the following parameters:

"returnURL": the URL to return the results to using an HTTP GET. The URL MUST NOT contain query parameters; this simplifies passing the results back as query parameters. This parameter should be used when the App is invoking the Agent directly.
"callbackURL": the URL to return the results to as a POST. The results are passed as a JSON object. This parameter should be used when the App is invoking the Agent indirectly through the scan of a QR code.

The "auth" object MAY contain the following properties:

"passcode": a flag indicating if the User must enter their Passcode. Default is true.
"authorization": – a flag indicating if the User must be prompted to authorize the transaction. Default is true.

Additional authentication parameters may be agreed to between Authentication Servers and Resource Servers to communicate other meta data about how the User MUST be authenticated.

Example Agent Request:

```
{ "iat": 1300819380
, "iss": "app.example.com"
, "aud": "ix.ca"
, "request.a2p3.org":
    { "resources":
        [ "people.bc.ca/details"
        , "health.bc.ca/number"
        ]
    , "auth":
        { "passcode": true
        , "authorization": true
        }
    , "returnURL": "https://app.example.com/return"
    }
}
```

8.2. Identifier Exchange Request

The "aud" property will be the Identifier Exchange ID and the signing key and key id will be the one obtained from the Registrar. The "request.a2p3.org" object MUST contain the following properties:

"token": the IX Token returned by the Agent
"request": the original Agent Request

Example IX Request payload:

```
{ "iat": 1300819542
, "iss": "app.example.com"
, "aud": "ix.ca"
, "request.a2p3.org":
    { "token": "h9gVsb … kU8C3lk"
    , "request": "lkj3hBv … Ui90bN"
    }
}
```

8.3. Resource Server Request

The "aud" property will be the Resource Server ID and the signing key and key id will be the one obtained from the Resource Server. The "request.a2p3.org" object MUST contain the following property:

"token": the RS Token returned by the Identity Exchange

Example RS Request payload:

```
{ "iat": 1300819542
, "iss": "app.example.com"
, "aud": "health.bc.ca"
, "request.a2p3.org":
    { "token": "kjR31h … pL8h3lk" }
}
```

9. Tokens

Tokens are encoded per JWT[1] and encrypted per JWE[3]. All tokens have the same header.

Example Header:

```
{ "typ":"JWE"
, "alg":"dir"
, "enc":"A128CBC+HS256"
, "kid":"Hs3Wlkp"
}
```

All Token payloads MUST contain the following properties:

"iat": date/time stamp when Token was generated as IntDate
"iss": ID of issuer
"aud": ID of audience
"sub": User's Directed Identifier shared between the issuer and the audience
"token.a2p3.org": object containing properties about the token

meta data about the User authentication and authorization

The token.a2p3.org MUST contain an `auth` object that MUST contain the following properties:

"passcode": flag indicating if the User entered their Passcode
"authorization": flag indicating if the User explicitly authorized the transaction

The auth object MAY contain:

"nfc": flag indicating if the User used an NFC card
any other properties as agreed to between the AS and RS to indicate how the User was authenticated.

9.1. IX Token

The IX Token is created by the Authentication Server and is consumed by the Identifier Exchange.

The “aud” property will be the Identifier Exchange ID.
The “iss” property will be the Authentication Server ID.
The encrypting key and key id will be the one shared between the IX and AS.

The “token.a2p3.org” object MUST also contain the following property:

“sar”: the signature of the Agent Request

Example IX Token payload:

```
{ “iat”: 1300818210
, “iss”: ”as.example.com”
, “aud”: ”ix.ca”
, “sub”: ”7uH2bnGD”
, “token.a2p3.org”:
    { “auth”:
        { “passcode”: true
        , “authorization”: true
        }
    , “sar”: “jB6shje12Gdsjhasdkjaoiqwlasd”
    }
}
```

9.2. RS Token

The RS Token is created by the Identifier Exchange and is consumed by a Resource Server.

The “aud” property will be the Resource Server ID.
The “iss” property will be the Identifier Exchange ID.
The encrypting key and key id will be the one shared between the IX and RS.

The token.a2p3.org object MUST contain the following properties:

“app”: the App ID that was granted authorization by the User
“scopes”: an array of Resource Authorization URLs granted by the User

“auth”: the auth object that was contained in the IX Token

Example RS Token payload:

```
{ “iat”: 1300813180
```

```
, “iss”: ”ix.ca”
, “aud”: ”health.bc.ca”
, “sub”: ”8udh36fja”
, “token.a2p3.org”:
    { “auth”:
        { “passcode”: true
        , “authorization”: true
        }
    , “app”: “app.example.com”
    , “scopes”: [“health.bc.ca/number”]
    }
}
```

10. APIs

10.1. API Calling Conventions

Except where noted, all API calls are made by making a POST request to the API URL of type "application/x-www-form-urlencoded" with the one parameter of "request" having the value of a Request string. All requests MUST use HTTPS.

Except where noted, all API calls return a JSON object, which MUST contain the "result" property if the call successfully returns results, "success" if no results and call was successful, or "error" object if the call failed. The "error" object contains the property "code" and an optional "message" property to provide additional context to a Developer.

10.2. Agent API

Unlike all the other APIs, the Agent is invoked by calling using the "a2p3" scheme rather than than "https". The Agent MUST register support with the platform for the "a2p3" scheme. Parameters and results are passed as query parameters to the URL. The API is called by creating a URL with the "a2p3" scheme, the API path and the parameters and then loading the URL on the device the User is interacting with. Alternatively, the API is called by creating a QR code that is scanned by the User with their Personal Agent.

/info

NOTE: non-standard API
IMPLEMENTATION NOTE: currently not implemented
Called by: App
Purpose: provides information about the Agent
Parameters:
"returnURL": URL to send the response
Returns:
"version" version of A2P3 supported. "0.0.1" is only valid response.
"name" display name of the Agent

/token

NOTE: non-standard API
Called by: App
Purpose: initiates the Agent requesting authentication and authorization from the User of the Agent Request
Parameters:
"request"
"state": an optional parameter for the App to preserve state
Returns: (these are returned on the query string if "returnURL" was in the Agent Request, otherwise they are properties in the JSON message if "callbackURL" was specified:
"token": IX Token if successful,
"request": the original Agent Request,

"state": the state parameter if provided by the App
"error": the error code if a request was not successful
"errorMessage": a message about the error

Error codes:

"USER_CANCELLED": The User cancelled the transaction.
"INVALID_APP_ID": Registrar did not recognize the App ID.
"INVALID_REQUEST": The Request structure or signature is invalid.

Example URL:

```
a2p3://token?state=k3jdh76h&request=jhysu…8U7lqcjhy
```

where `jhysu…8U7lqcjhy` is an Agent Request

QR Code Example:

The App generates a QR code for:

```
https://app.example.com/qr/kdsuyehjscaskjdset
```

The Personal Agent reads the QR code and detects the URL does not have the "a2p3" scheme, appends "&json=true" to the Short URL and receives a JSON response containing the "request" and optional "state" parameter

10.3. Registrar API

/request/verify

NOTE: non-standard API, Agent does a POST call with the "request" and "token" parameters. Standard JSON response format.

Called by: Personal Agent

Purpose: checks if an Agent Request from an App is valid

"request": Agent Request received from App

"token": agent token

"result":

"name": App name that was registered at the Registrar

Error codes:

"INVALID_APP_ID": Unknown App ID in Agent Request.
"INVALID_REQUEST": The Agent Request structure or signature is invalid.
INVALID_TOKEN": agent token is invalid

/report

NOTE: non-standard API, Agent does a POST call with the "request" and "token" parameters. Standard JSON response format.

Called by: Personal Agent

Purpose: called when User reports an App has made a suspicious authorization request.
Parameters:
"request": the Agent Request the App sent the Personal Agent
"token": agent token
"result":
"success":true if successful
Error codes:
"INVALID_APP_ID": Unknown App ID in Agent Request.
"INVALID_REQUEST": The Agent Request structure or signature is invalid.
INVALID_TOKEN": agent token is invalid

/authorizations/requests

NOTE: non-standard API, Agent does a POST call with a JSON object as the body. Standard JSON response format.
Called by: Personal Agent
Purpose: gets RS Requests from the Registrar so the Personal Agent can call Resource Servers "/authorizations/list" API
JSON body:
"authorizations": an array of Resource Servers
"token": agent token
"result":
An object with each property being the RS ID containing the RS Request
Error codes:
INVALID_TOKEN": agent token is invalid

/app/verify

Called by: Resource Server
Purpose: verifies if the User in the RS Token for the Registrar has administrative authority for an App ID
"request.a2p3.org":
"token": RS Token for the Registrar
"app": App ID the Developer would like to register at the Resource Server
"result":
"name": App name that was registered at the Registrar
Error codes:
"INVALID_APP_ID": Unknown App ID.
"UNAUTHORIZED": The User is not authorized to administer the App ID.

/app/list

Called by: Resource Server

Purpose: provides a list of all App IDs that the User in the RS Token for the Registrar has administrative authority for

"request.a2p3.org":

"token": RS Token for the Registrar

"result":

An object where each property name is the App ID, and the value is the name the App was registered under.

Error codes:

"UNKNOWN_USER": The User does not have any registered Apps.

10.4. Authentication Server API

/agent/delete

Called by: Identity Exchange

Purpose: called when User has asked to deactivate a given Personal Agent

"request.a2p3.org":

"handle": Personal Agent Handle provided to the IX then Personal Agent was enrolled

"result":

"success":true if successful

Error codes:

"INVALID_HANDLE": Unknown Personal Agent Handle.

10.5. Identifier Exchange API

/di/create

Called by: an Enrollment App

Purpose: creates a new User

"request.a2p3.org":

"AS": Authentication Server ID

"RS": an array of RS IDs

"redirects": an object mapping Standard Resource IDs to arrays of RS IDs

"result":

"dis": an object mapping the AS ID and any RS IDs to their Directed Identifier for the User

/exchange

Called by: Apps

Purpose: generates RS Tokens per IX Token and Agent Request

Parameters: an IX Request

"result":

"sub": the directed identifier for the User for the calling App

"tokens": an object mapping App ID to RS Token

"redirects": an object mapping any Standardized Resources to

an array of zero or more App IDs for Resource Servers that support the Standardized Resource for the User

Error codes:

"INVALID_TOKEN": IX Token is not valid
"INVALID_IXREQUEST": IX Request is invalid
"EXPIRED_IXREQUEST": IX Request has expired
"INVALID_REQUEST": Agent Request is invalid
"EXPIRED_REQUEST": Agent Request has expired
"UNKNOWN_RESOURCE": A Resource Server ID is unknown. Details are contained in the message property

/agent/list

Called by: any Authentication Server
Purpose: provides a list of Personal Agents for the User identified in the RS Token for the IX
"request.a2p3.org":
"token": RS Token for IX
"result":
"agents": and array of objects containing properties;
"display": display name for Personal Agent
" handle": handle used to delete Personal Agent
Error codes:
"INVALID_TOKEN": RS Token is not valid.

/agent/add

Called by: any Authentication Server
Purpose: adds a Personal Agent for the User identified in the RS Token for the IX
"request.a2p3.org":
"token": RS Token for IX
"handle": opaque handle provided by AS to identify the User Personal Agent later
"result":
"token": opaque agent token if successful, used by agent to make calls to Registrar
Error codes:
"INVALID_TOKEN": RS Token is not valid.
"DUPLICATE_HANDLE": Personal Agent Handle already exists.

/agent/delete

Called by: any Authentication Server
Purpose: deletes a Personal Agent for the User identified in the RS Token for the IX

“request.a2p3.org”:

“token”: RS Token for IX

“handle”: opaque handle provided previously by AS to identify the User Personal Agent

“result”:

“success”:true if successful

Error codes:

“INVALID_HANDLE”: Unknown Personal Agent Handle.

10.6. Resource Server API

If the Resource Server is part of the enrollment process, it has the following API to link an existing User record to an A2P3 DI:

/di/link

Called by: Enrollment App
Purpose: links an identifier known to the RS for a given User to a DI during User enrollment
"request.a2p3.org":
"identifer": existing identifier RS has for User
"di": A2P3 directed identifier for RS for User
"result":
"success":true if successful
Error codes:
"INVALID_REQUEST": Invalid RS Request
"INVALID_IDENTIFIER": Unknown RS identifier

If the Resource Server grants Anytime access to any of its resources, it needs to have indicated that upon app registration and support the following APIs:

/authorizations/list

NOTE: Non-standard API. The Request is generated by the Registrar even though call is made by a Personal Agent
Called by: Personal Agent
Purpose: lists all Apps and Resources that have been authorized by User
"result":
An object where each property is an App ID that contains:
"name": name of App
"lastAccess": time of last access of App
"resources": array of authorized resource URLs
"request": an RS Request the agent can use to delete the authorization
Error codes:
"INVALID_TOKEN": Invalid RS Token

/authorization/delete

NOTE: Non-standard API. The RS Request is provided by the RS in its response to "/authorizations/list"
Called by: Personal Agent
Purpose: deletes all Resource authorizations for the App ID contained in the RS Request.
"result":
"success":true if successful
Error codes:
"INVALID_REQUEST": Invalid RS Request
"INVALID_APP": Unknown App for User

11. Resource Descriptions

The resource description is shown to the User so that the User is informed of what the App is requesting of the Resource. The result is a JSON object of each language supported.
The string is in Markdown (http://daringfireball.net/projects/markdown/syntax).
NOTE: MD allows embedded HTML, is this the best choice?

Example resource description:

```
{“en”:”Access your full name and photo from people.bc.ca”}
```

12. Standardized Resources

Some Resource Servers provide identical identity resources and standardizing the API and Resource Authorization Descriptions would be useful to Users and App Developers. These Standardized Resources are at a host name that provides documentation for the common API and hosts the standard Resource Authorization Descriptions.

Some Apps will not know which Resource Server has resources for a given User. Resource Servers providing a Standard Resource can register which User they provide the Standard Resource for. When Apps ask for authorization for a Standard Resource, the Identifier Exchange will return an array of Resource Servers that provide the Standardized Resource for the User.

For example, provincial health Resource Servers could have a standardized API and publish the API and a Resource Authorization Description at “https://health.ca/prov_number”. A User can then be is added to the system and “health.bc.ca” could be added as a Resource Server for the User that supports the Standardized Resource “health.ca”. When an App asks for “health.ca/prov_number”, the Identifier Exchange will return to the App the Resource Server “health.bc.ca” that provides the Standardized Resource, “health.ca”.

This capability helps a set of Resource Servers simplify the process the App must go through to resolve which Resource Server within the set the App must use for a given User. It also simplifies the User experience at the App by avoiding the requirement to present the User with the full set of Resource Servers to choose among (when in fact the User only deals with one, or a subset, of the total set of Resource Servers).

13. Security Considerations

13.1. The App's signing keys MUST be kept secret and in the server component of the App.

13.2. The interactions between the App and the Agent can easily be intercepted and observed.

13.3. Does the Return URL for the Agent Request need to be registered at the Registrar to mitigate App impersonation?

13.4. To Be Continued

14. Privacy Considerations

14.1. User data MUST only be released by Resource Servers to an App when the User has been presented with the data release description and authorized the transaction. Privileged Apps that have been verified by the Resource Server or a proxy and that are acting in the best interests of the User or according to legislation may obtain User data without User authorization.

14.2. A2P3 does not provide colluding Resource Servers a common identifier, but if the User shares a common identifier such as an email address or photo, colluding Resource Servers may then be capable of linking the user activity.

14.3. Colluding Resource Servers may statistically link a user by sharing transaction timing if both parties are part of a common transaction.

14.4. Resource Servers SHOULD provide a means for a User to see which Apps have access to which resources.

14.5. Possible User resource authorization management technique. An API at Resource Servers for User's to query to determine which resources have been granted to which Apps and API to delete specific authorizations. A Personal Agent could remember which Resources have been requested and allow User to query all Resources and manage authorizations.

14.6. To Be Continued

15. References

15.1. Normative References

1. JSON Web Token (JWT) http://tools.ietf.org/html/draft-jones-json-web-token-10
2. JSON Web Signature (JWS) http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-06
3. JSON Web Encryption (JWE) http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-06
4. JSON
5. Language encodings http://tools.ietf.org/html/rfc5646

15.2. Informative References

1. HTTP protocol
2. JSON
3. QR Codes
4. URI Schemes http://www.iana.org/assignments/uri-schemes.html
   http://www.quora.com/Where-can-one-find-an-aggregate-list-of-all-protocol-handlers-that-all-apps-in-the-iTunes-App-Store-register-with-iOS
5. X-callbadk-url 1.0 http://x-callback-url.com/specifications/

Appendix A – Transaction and Identifier Containment Matrix

Privacy Protection

The following table illustrates which parties see the Authorization Request and which user identifiers:

	App	Agent	Registrar	AuthN Service	Identifier Exchange	Resource Server
AuthZ Request	visible	visible	visible	never sees	visible	never sees
AuthN Identifier	never sees	visible	never sees	visible	never sees	never sees
Passcode	never sees	visible	never sees	visible	never sees	never sees
AS User Identifer	opaque	opaque	never sees	visible	visible	never sees
RS User Identifer	opaque	never sees	never sees	never sees	visible	visible
Public RS User Identifier	may see	never sees	never sees	never sees	never sees	visible

Appendix B – App Flow Example

TBD

Appendix C – Acknowledgements

Many people contributed to this document. This specification would not have happened without the determination of Peter Watkins and his merry band of Patricia Wiebe and Ian Bailey under the guidance of Dave Nikolejsin. This document is derived from the GoBC Identity Architecture funded by SecureKey with contributions from Hugh Cumming, Troy Ronda and Mike Varley. The protocol was presented at Internet Identity Workshop XV where substantial feedback was provided.

Appendix D – License

As of [insert date], the following have executed Contributor License Agreements and made this Specification available under the Open Web Foundation Contributor License Agreement Version 1.0:

- Dick Hardt
- Province of British Columbia
- SecureKey Technologies Inc.

You can review the signed copies of the Open Web Foundation Contributor License Agreement Version 1.0 for this Specification at: http://www.a2p3.net/

Your use of this Specification may be subject to other third party rights. THIS SPECIFICATION IS PROVIDED "AS IS." The contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to the Specification. The entire risk as to implementing or otherwise using the Specification is assumed by the Specification implementer and user. IN NO EVENT WILL ANY PARTY BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Document History

Draft 1, Oct 21, 2012: incorporated GoBC Architecture Document and numerous changes.
Draft 2, Nov 7, 2012: incorporated feedback from IIW XV, added resource redirect.
Draft 3, Nov 11, 2012: wide variety of completion tasks.
Draft 4, Nov 12, 2012: fleshed out API calls and Error Codes.
Draft 5, Nov 15, 2012: incorporated feedback from Peter Watson and Greg Turner
Draft 6, Nov 16, 2012: minor edits
Draft 7, Nov 20, 2012: minor edits
Draft 8, Nov 23, 2012:
- changed “Mobile Agent” to “personal agent”
- added Agent Token
- changed Resource Server /authorization/* calls to come from Agent
- added /report API to Registrar

Draft 9, March 8, 2013:
- Changed Agent scheme to be “a2p3”
- Changed all Agent calls to send and receive JSON
- Changed how redirect results are returned
- Removed Copyright and added License Appendix

Draft 10, Oct 6, 2013:
- Added explanation of public identifiers to Sec. 2 Privacy Model
- Added /di/link API to Sec. 10.6
- Removed subsequent PC Web App flow from Sec. 6
- Removed references to Notification URL