mitrelogo-verysmall-white.png

Sponsor: Department of Veterans Affairs

Contract No.: VA791-P-0042

Project No.: 40144028

**Document No: MTR140188**

**McLean, VA**

Secure RESTful Interfaces: Business-oriented Use Cases & Associated Distributed Security Requirements

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official government position, policy, or decision, unless designated by other documentation.

This document was prepared for authorized distribution only. It has not been approved for public release.

©2014 The MITRE Corporation. All rights reserved.

M. Russell

May 2014

This page intentionally left blank.

Table of Contents

[1. Introduction 1](#_Toc388019948)

[1.1. Task Background 1](#_Toc388019949)

[1.2. Introduction to REST 2](#_Toc388019950)

[1.3. REST in the Health IT Realm 3](#_Toc388019951)

[2. Scope and Assumptions 3](#_Toc388019952)

[3. Open Security Standards for RESTful Interfaces 4](#_Toc388019953)

[3.1. Transport Layer Security (TLS) 4](#_Toc388019954)

[3.2. OAuth 2.0 5](#_Toc388019955)

[3.3. JWT 5](#_Toc388019956)

[3.4. JOSE 5](#_Toc388019957)

[3.5. OpenID Connect 1.0 6](#_Toc388019958)

[4. Use Cases and Patterns for Secure RESTful Interfaces 6](#_Toc388019959)

[4.1. Client Delegation Pattern 7](#_Toc388019960)

[4.1.1. Description 7](#_Toc388019961)

[4.1.2. Specific Use Case Examples 7](#_Toc388019962)

[4.1.3. Initiation 7](#_Toc388019963)

[4.1.4. Use Case Actors 7](#_Toc388019964)

[4.1.5. Assumptions and Prerequisites 8](#_Toc388019965)

[4.1.6. General Sequence of Events 8](#_Toc388019966)

[4.1.7. Post-conditions 10](#_Toc388019967)

[4.1.8. Security Requirements 10](#_Toc388019968)

[4.2. Identity Federation (VA as Relying Party) Pattern 12](#_Toc388019969)

[4.2.1. Description 12](#_Toc388019970)

[4.2.2. Specific Use Case Examples 12](#_Toc388019971)

[4.2.3. Initiation 13](#_Toc388019972)

[4.2.4. Use Case Actors 13](#_Toc388019973)

[4.2.5. Assumptions and Prerequisites 13](#_Toc388019974)

[4.2.6. Sequence of Events 13](#_Toc388019975)

[4.2.7. Post-conditions 15](#_Toc388019976)

[4.2.8. Security Requirements 15](#_Toc388019977)

[4.3. Identity Federation (VA as OpenID Provider) Pattern 16](#_Toc388019978)

[4.3.1. Description 16](#_Toc388019979)

[4.3.2. Specific Use Case Examples 16](#_Toc388019980)

[4.3.3. Initiation 17](#_Toc388019981)

[4.3.4. Use Case Actors 17](#_Toc388019982)

[4.3.5. Assumptions and Prerequisites 17](#_Toc388019983)

[4.3.6. General Sequence of Events 17](#_Toc388019984)

[4.3.7. Post-conditions 19](#_Toc388019985)

[4.3.8. Security Requirements 19](#_Toc388019986)

[4.4. Potential Future Pattern – User-Managed Access 19](#_Toc388019987)

[5. Next Steps 22](#_Toc388019988)

[List of Acronyms 23](#_Toc388019989)

[References 25](#_Toc388019990)

List of Figures

[Figure 1 – Client Delegation Pattern 7](#_Toc388019991)

[Figure 2 - OAuth 2.0 Authorization Code Flow 9](#_Toc388019992)

[Figure 3 – Identity Federation (VA as RP) Pattern 12](#_Toc388019993)

[Figure 4 – OpenID Connect Authorization Code Flow 14](#_Toc388019994)

[Figure 5 – Identity Federation (VA as OP) Pattern 16](#_Toc388019995)

[Figure 6 – OpenID Connect Authorization Code Flow 18](#_Toc388019996)

[Figure 7 - UMA Architecture 20](#_Toc388019997)

[Figure 8 - UMA Core Protocol Flow 21](#_Toc388019998)

List of Tables

[Table 1 - Client Delegation Pattern Use Case Actors 8](#_Toc388019999)

[Table 2 – Identity Federation (VA as RP) Use Case Actors 13](#_Toc388020000)

[Table 3 – Identity Federation (VA as OP) Use Case Actors 17](#_Toc388020001)

# Introduction

This document identifies use cases and security requirements for Secure Representational State Transfer (RESTful) Interfaces in support of the development of a Secure RESTful Interfaces Profile for the Department of Veterans Affairs (VA) Deputy Chief Information Officer, Office of Information and Technology (OIT), Architecture, Strategy, and Design (ASD).

## Task Background

The MITRE Corporation’s work program in support of VA ASD is organized by VA outcomes. One of the ASD outcomes is *Integration with Mission Partners*. The Integration with Mission Partners outcome statement is: *The VA interoperates seamlessly with mission partners enabled by architectural alignment*. This Outcome is focused on VA interactions with external entities, including:

* The US Department of Defense (DoD)
* Other Federal agencies
* Health care providers
* Other commercial organizations
* Veterans, Veteran Support Organizations, and Caregivers
* The general public

MITRE’s 2014 support to the Integration with Mission Partners outcome is divided into Task Areas. The *Modern Open Architecture* Task Area, which includes work on Secure RESTful Interfaces, provides technical contributions to accelerate VA’s use of open standards to facilitate mission partner interoperability.

This work is based on the premise that adopting open standards that are in wide use on the commercial web today, such as the RESTful architecture style (discussed in Section 1.2), will enable VA to integrate with external partners across all of the categories listed above with less expense and effort and with greater reuse of deployed systems and interfaces. These architectural changes are necessary to support Strategic Objective 3.2 of the VA Strategic Plan: “Evolve VA Information Technology Capabilities to Meet Emerging Customer Service / Empowerment Expectations of Both VA Customers and Employees” [1].

Growing interest in REST is evident at the VA and among its Federal and commercial mission partners, as well as the wider Health Information Technology (IT) community. Given the VA’s responsibility to safeguard information, including the private health information of the Nation’s veterans, it is incumbent on the VA to adopt new technologies in a secure, reliable, and trustworthy manner. MITRE’s work on Secure RESTful Interfaces offers technical guidance in support of this objective. Therefore the focus of this MITRE project is not on specific RESTful interfaces themselves, but rather how they can be secured using open security standards. Much of the work will entail the definition of profiles for these standards to meet the VA’s security requirements.

This document describes a set of common REST security patterns, such as client delegation and identity federation, and identifies specific use case examples and security requirements for each pattern. These patterns and use cases will form the basis for defining guidelines and standards profiles for secure RESTful interfaces. After the completion of these deliverables, a second phase of the project will select a pilot use case and produce a working pilot implementation to demonstrate the viability of REST interfaces built using the proposed guidelines and profiles.

## Introduction to REST

While a full description of REST is beyond the scope of this document, the following brief introduction may provide some useful background.

REST is a lightweight architectural style that exploits the full functionality and semantics of the Hypertext Transport Protocol (HTTP) for client-server applications [2]. Some key elements of REST include:

* Individual resources are uniquely addressable by Uniform Resource Locators (URLs)
* Actions such as creating, updating, and deleting resources are performed using the HTTP verbs (e.g., GET, PUT, POST, DELETE)
* Stateless interactions are supported, with no dependency on session continuity between requests
* Hypermedia Controls, such as link relations, are used to describe the relationships among resources and communicate other available resources to the client

REST is often contrasted with SOAP[[1]](#footnote-1), an Extensible Markup Language (XML)-based message protocol that is typically transmitted over HTTP, and often used in Service-Oriented Architectures (SOA). SOAP provides a mechanism for wrapping arbitrary data in a standard message format, which is typically transmitted over HTTP to a designated endpoint URL. A single SOAP endpoint URL may process many different types of messages, including requests to query, update, or delete data. Therefore requests that appear very similar at the HTTP layer, perhaps using the same URL and HTTP verb, may carry embedded messages that perform a variety of different functions and access different resources. In order to understand the request, the message payload must be extracted from the SOAP envelope, and parsed according to the specifications of its format. In addition, SOAP exchanges often make use of the XML Digital Signature and XML Encryption standards to provide message-level protection of message confidentiality, authenticity, and integrity.

In a REST design, the application semantics are implemented directly in the HTTP protocol. Typically, the resource to be accessed is indicated by the URL, the action to be taken is indicated by the HTTP verb, and further parameters are specified in the URL query string or request body. REST interfaces commonly forego message-level encryption and signatures, relying on Transport Layer Security (TLS) to protect message integrity and confidentiality in transit. It should be noted that while XML and XML-based standards such as the Security Assertion Markup Language (SAML) are typically associated with SOAP, they can be used in RESTful exchanges as well. However, the simpler JavaScript Object Notation (JSON) standard is much more widely used in REST implementations.

While SOAP remains a common choice for enterprise SOA implementations, REST is the predominant choice for public web Application Programming Interfaces (APIs). In 2010, ProgrammableWeb, an online web API directory, found that the percentage of its listings that were SOAP-based fell from 25% to 15% between 2008 and 2010. Over the same period, the REST-based share of its listings increased from 60% to 74% [3]. Proponents of REST claim that RESTful interfaces can greatly simplify client application development, especially for services that must accommodate different types of clients including mobile and browser-based JavaScript clients.

## REST in the Health IT Realm

As medical organizations adopt a range of Electronic Health Record (EHR) systems with various and often incompatible data formats, numerous efforts have been launched to provide standard mechanisms for the exchange, integration, and sharing of electronic health information. Health Level 7 (HL7), an international non-profit health IT standards organization, produces standards widely used for medical data exchange. HL7’s hData standard and the subsequent draft Fast Health Interoperable Resources (FHIR) [4] standard support RESTful access to health data through a modular scheme of addressable resources such as “patient” or “condition.”

A June 2013 presentation by the Office of the National Coordinator for IT Health (ONC) Health IT Standards Committee concluded that the combination of secured RESTful transport (HTTP Secure, or HTTPS), OpenID Connect authentication, OAuth 2 authorization, and FHIR healthcare content constitutes “a safe and appropriate set of standards to use as building blocks for more complicated healthcare applications” [5]. This presentation referenced two key related initiatives – the RESTful Health Exchange (RHEx) [6] and BlueButton+ (BB+) Pull [7] pilots – that demonstrated the use of REST and open standards to support secure health information exchange. The report recommended that ONC “support and encourage the development and piloting of BB+, FHIR, and RHEx.” This task seeks to build upon these and other related efforts.

Currently, there are multiple efforts underway to provide RESTful interfaces to the Veterans Health Information Systems & Technology Architecture (VistA), the VA’s EHR system, using FHIR as the basis for a RESTful API. ASD also supports the adoption of REST interfaces in support of its goals to reduce the cost and complexity of developing client software applications. VA is poised to adopt REST for a number of new initiatives, and it is likely to remain a prevalent design philosophy at the VA for some time to come.

# Scope and Assumptions

The following paragraphs state the underlying assumptions and scope boundaries that will guide the Secure RESTful Interfaces work.

1. Focus on External Interfaces

The Secure RESTful Interfaces task is focused on interfaces between the VA and external entities, including the DoD, other Federal agencies, health care providers and other commercial entities, veterans and caregivers, and the general public. Interfaces intended for internal VA consumption are out of scope.

1. Present a Representative Sample of Use Cases

This document presents a representative sample of use cases for secure RESTful interfaces. It is not a comprehensive list, nor is it meant to constrain the set of use cases to which a secure RESTful approach could be applied.

1. Include Use Cases from Multiple Domains

The scope of this task is not restricted to the health care domain. Given the VA’s mission, health care is clearly a major area of focus, but this task’s goal is to produce security guidance that is applicable across multiple domains.

1. Use Open Standards

A core tenet of ASD’s architecture strategy is the adoption of open standards to maximize interoperability and reduce total cost of ownership of applications. The Secure RESTful Interfaces Profile will use open standards in support of this goal.

1. Provide Interface-Agnostic Guidance

In order for the security guidance produced through this task to be applicable across multiple domains and interfaces, it must necessarily be agnostic to the specific content of any given REST API. At the same time, security guidance must acknowledge that APIs will handle data with varying levels of sensitivity, and different business processes will need to address different risks and threats. Where appropriate, the guidance provided will include options that may be selected based on the needs of a particular API and implementation.

1. Take a Forward-Looking Approach

The guidance provided will aid VA in moving toward its desired future-state architecture. In addition to well-established standards and technologies that are available today, the guidance produced for this task will also consider the potential roles of emerging standards and technologies.

# Open Security Standards for RESTful Interfaces

The following sections briefly introduce some of the open security standards for securing RESTful web interfaces. The guidance presented in the Secure RESTful Interfaces Profile will largely consist of the application of these standards with specific constraints in order to obtain the required security assurances.

## Transport Layer Security (TLS)

TLS is an Internet Engineering Task Force (IETF) standard for securing a communications channel between a client and a server [8]. A TLS exchange begins with a series of handshake messages in which the server (and optionally the client) is authenticated using X.509 certificates, and a symmetric session key is established. TLS provides transport-layer encryption, integrity protection, and authentication. Many application protocols can be sent over a TLS-secured channel, including HTTP in the form of HTTP Secure (HTTPS). TLS 1.2 is the most recent version of the protocol, but use of it is not yet ubiquitous, with many software packages supporting only TLS versions 1.1 and below.

## OAuth 2.0

OAuth 2.0 is an IETF standard (Request for Comment or RFC 6749) [9] that enables a resource owner to authorize delegated access by a third-party client to protected resources. Several other RFCs provide extensions to the core specification.

Client delegation eliminates the need for resource owners to provide third-party clients with their own credentials, a practice which allows the client to impersonate the owner and perform any actions to which the owner is authorized. The client delegation approach enables the owner to manage the client’s credentials, revoking them when appropriate, and to limit the scope of authorization.

OAuth 2.0 incorporates a number of different protocol flows and a wide range of optional features, and is therefore often referred to as an “authorization framework” rather than a protocol. The “looseness” of the OAuth 2.0 specification enables a tremendous variability in OAuth 2.0 implementations with no guarantee of interoperability among them and vastly different security properties. On the other hand, the adaptable OAuth framework has provided the basis for other useful protocols serving different but related purposes, including OpenID Connect and User-Managed Access (UMA).

For VA to adopt OAuth 2.0, it will be necessary to define a profile to constrain the set of options available to implementations and provide the required security. The Secure RESTful Interfaces Profile team will review and update the profiles created for the RHEx and Blue Button+ Pull projects to produce the profiles for this task.

## JWT

JSON Web Token (JWT) is one of several standards based on JSON, a lightweight format for expressing data objects as a set of name-value pairs using human-readable text. JWT is a draft IETF standard for conveying a set of claims in a JSON object [10], where each claim is a name/value pair. The JWT standard defines a set of standard claims, and establishes a public registry where claim names can be adopted and published for widespread public use. Examples of the standard JWT claims include “iss” (issuer), “sub” (subject), and “exp” (expiration time).

Arbitrary claim names can also be used, though it is then incumbent on the producer of the claims to agree on their meanings with consumers of the claims. JWTs can be signed or encrypted using the JSON Object Signing and Encryption (JOSE) specifications.

## JOSE

JOSE is a set of IETF draft standards to enable cryptographic protection of JSON objects. The individual standards are:

* JSON Web Signature (JWS) – JSON objects with digital signatures or Message Authentication Codes (MAC) [11]
* JSON Web Encryption (JWE) – Encrypted JSON objects [12]
* JSON Web Keys (JWK) – public and private keys (or sets of keys) represented as JSON objects [13]
* JSON Web Algorithms (JWA)– specifies cryptographic algorithms to be used in the other JOSE standards [14]

JOSE performs functions similar to the XML Signature and XML Encryption standards, though there are some notable differences. The XML cryptographic standards depend on the use of X.509 certificates issued by trusted Certificate Authorities (CAs). JWK supports the use of X.509 certificates, but does not require it. XML Signatures also require the XML document to be converted into a standard canonical form before calculating the signature, but JWS takes the simpler approach of signing a JSON object in the exact form in which it will be sent across the network.

## OpenID Connect 1.0

OpenID Connect 1.0 defines a standard RESTful interface for identity federation, enabling an application (referred to as the Relying Party or RP in this context) to receive authenticated user identity information from a trusted OpenID Provider (OP). The OpenID Connect standard [15] is built on OAuth 2.0 and managed by the OpenID Foundation. OpenID Connect also uses the draft JWT standard to provide a format for its ID Tokens.

OpenID Connect is analogous to SAML: OPs are comparable to SAML Identity Providers (IdPs), and the claims conveyed in the OpenID Connect ID Token are similar to the Attribute elements passed in a SAML assertion. The OpenID Connect standard defines a set of required and optional claims to be included in the ID Token, and OPs may include any additional claims as needed. Claims can be used to convey the user’s identifier, attributes, and information about how the user was authenticated, such as the type of credentials used. OpenID Connect requires that ID Tokens be signed using JWS, and they may also be encrypted using JWE.

# Use Cases and Patterns for Secure RESTful Interfaces

Given the broad scope of the Secure RESTful Interfaces task, its specifications must necessarily be somewhat general and agnostic of the subject domain, data content, and participants involved in specific use cases. At the same time, the guidance must address real business needs facing VA in its interaction with mission partners in order to be relevant. In an attempt to balance these concerns, this document presents a set of patterns, which are common RESTful security flows at an abstract level. For each pattern, a set of concrete and specific VA use cases are identified.

Relevant security requirements for each pattern are identified and described. These patterns and requirements will form the basis for analysis and definition of security guidance and profiles that will make up the final products of the Secure RESTful Interfaces task.

The following sections describe each use case pattern, specific use cases that are instances of the pattern, the actors involved and flow of events, and security requirements pertaining to the pattern. Three patterns are described – client delegation, and two identity federation patterns (one with VA as the OP and one with VA as the RP). User-Managed Access is also introduced as a potential pattern for future consideration.

## Client Delegation Pattern

### Description

A Resource Owner chooses to grant a third-party software client delegated access to a protected resource – the canonical use case for which OAuth was created. In this pattern, the protected resource resides on a VA system.

Figure 1 below provides a functional depiction of this use case. This figure does not represent the actual protocol message flow among participants, since this would depend on the specific OAuth flow used (e.g., Authorization Code vs. Implicit).

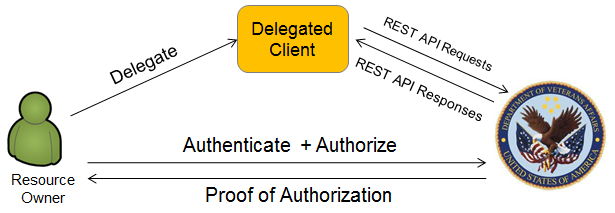


Figure – Client Delegation Pattern

### Specific Use Case Examples

Specific examples of the Client Delegation pattern include:

* A veteran delegates access to his/her EHR to a web application for tracking blood pressure
* A veteran delegates access to his/her EHR to a mobile application, limiting the scope of access to information pertaining to a particular medical condition
* A veteran uses a personal health monitoring device which uploads measurement data to an external web application; the veteran delegates access to the web service to upload this Patient-Generated Data (PGD) to a VA system

In any of these cases, the Resource Owner could instead be a Veteran Service Organization (VSO) or other authorized caregiver.

### Initiation

The Resource Owner initiates this use case by deciding to use a third-party client application to access resources residing at the VA. The VA may or may not have prior knowledge of the specific client application being used.

### Use Case Actors

Table 1 lists the actors involved in the Client Delegation pattern.

Table - Client Delegation Pattern Use Case Actors

| Actor | Description |
| --- | --- |
| Resource Owner | A user with authorized access to a protected resource residing on a Resource Server at VA. While this user may not be the direct “owner” of a resource, this user is authorized to make access decisions pertaining to the resource. For example, although a veteran is the owner of his/her EHR, a designated family member may be authorized to make decisions on the veteran’s behalf, and in that case could act as the Resource Owner of the EHR. |
| Client | A software application that can perform some function with the protected resource. For example, a blood pressure tracking application could access blood pressure readings from an EHR system to populate a database and present the data in some means that is useful to the Resource Owner. Many different types of clients exist with varying security properties. Examples of client types include hosted web applications, native desktop and mobile applications, and browser-based applications. |
| Authorization Server | An OAuth 2.0 Authorization Server which implements an Authorization Endpoint and a Token Endpoint. In this case, the Authorization Server is hosted by the VA. |
| Resource Server | The server that hosts the Protected Resource. In this case, the Resource Server is hosted by the VA. |

### Assumptions and Prerequisites

* The Resource Owner has a valid user account and credentials provisioned by the VA.
* The client is registered with the Authorization Server, has been issued client credentials, and has the URLs for the Authorization Server’s OAuth endpoints.
* The details of interactions between the Resource Server and Authorization Server have been established such that the Resource Server can validate and accept access tokens issued by the authorization server.
* The Resource Owner’s credentials will not be provided to the Client.

### General Sequence of Events

The following sequence is based on the OAuth Authorization Code flow. Other protocol flows with a different sequence of messages and different security properties are also defined in the OAuth specification. Selection of a specific flow and other parameters should take into account the security requirements of each particular use case.

The Authorization Code flow sequence is depicted in Figure 2 and described below. Using the use case example involving a veteran delegating EHR access to a web application for tracking blood pressure, the italicized text in the diagram indicates which role each of these parties would play in the protocol flow. The “user agent” referenced in the following description is a software client, commonly a web browser, that is under the direct control of the Resource Owner. The “Client” in OAuth terms is also a piece of software that performs some action on the Resource Owner’s behalf, but it may or may not be under the Resource Owner’s control. Clients may be native applications, hosted web applications, or applications that run directly in a web browser.

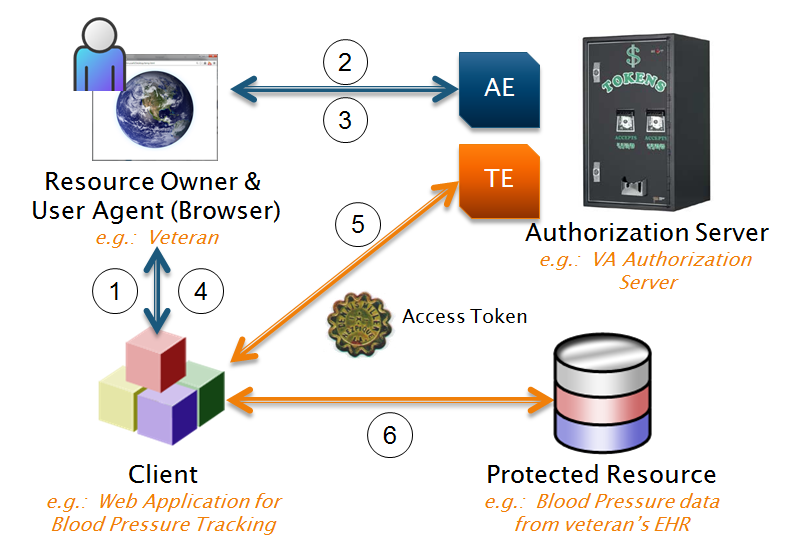


Figure - OAuth 2.0 Authorization Code Flow

1. The Resource Owner interacts with the Client and indicates a desire for the client to access the Protected Resource. The details of this process are client-specific.
2. The client redirects the Resource Owner’s user agent (a web browser) to the Authorization Server’s Authorization Endpoint (AE), which prompts the Resource Owner to authenticate with pre-established VA credentials, in accordance with VA policy. Note that no particular authentication mechanism is dictated by the OAuth standard or this pattern.
3. The Authorization Server prompts the Resource Owner to authorize the Client’s access to the Protected Resource. The Authorization Server should display available information about the request, including the requested scope of access, which can be specified in the optional *scope* parameter of the request.
4. The Authorization Server redirects the Resource Owner’s user agent back to the Client with one or more parameters pertaining to the outcome of the authorization request. If the request was denied, the Client is notified of this fact. If the request was granted, the client is provided with an Authorization Code which it can present to the Authorization Server as evidence of the owner’s authorization. The granted *scope* will also be returned to the client if it is different from what was requested.
5. The Client connects to the Authorization Server’s Token Endpoint (TE), using its client credentials to authenticate. The Client submits the Authorization Code. If the Authorization Server determines that the code is valid, it issues an Access Token to the client. Optionally, it may also issue a Refresh Token along with the Access Token. The Refresh Token can be used to obtain a new Access Token when the issued Access Token has expired. If the client does not have a Refresh Token and further access is needed after the Access Token expires, the client must re-initiate the authorization flow from the beginning.
6. The Client connects to the Resource Server, requests access to the Protected Resource, and presents the Access Token as proof of authorization. If the Resource Server determines that the Access Token is valid, it grants the Client access to the Protected Resource.

### Post-conditions

* The Client has an Access Token which it can present to the Resource Server as proof of authorization to access the Protected Resource.
* The Access Token has a limited lifetime defined by its *expires\_in* attribute, and optionally grants a subset of the authorizations available to the Resource Owner, as specified by the *scope* parameter.
* Optionally, if long-term authorization is required, the Authorization Server may issue a Refresh Token to the Client, which can be used to obtain a new Access Token when the issued Access Token has expired.

### Security Requirements

The OAuth specification and a companion document, OAuth 2.0 Threat Model and Security Considerations [16], contain several recommendations for securing OAuth implementations and preventing common implementation flaws. One example of an implementation flaw is the so-called Covert Redirect vulnerability discovered in May 2014 [17] and described as a flaw in the OAuth and OpenID Protocols. In fact, the Covert Redirect issue was not a flaw in the protocols, but the result of flawed client applications that allowed unrestricted redirection of requests to arbitrary sites, with parameters such as Authorization Codes attached. Section 10.15 of the OAuth 2.0 specification warns about open redirectors and their risk of exposing authorization codes – precisely the scenario described in the Covert Redirect vulnerability report. The issue would have been avoided if client implementers had read and followed the security guidance in the specification.

Significant security requirements pertaining to the client delegation pattern include:

* **Secure Communications Channels** – Resource Owner and Client Credentials, Access Tokens, and Authorization Codes all require confidentiality protection when transmitted over the network. In use cases where Protected Health Information (PHI) or other sensitive data is accessed, API requests and responses between the client and resource server will require transport encryption as well. Using TLS with server authentication also reduces the risk of man-in-the-middle (MITM) attacks.
* **Access Token, Refresh Token, and Authorization Code Lifetimes** – The OAuth provider can configure expiration parameters for each of these objects. Short-lived tokens and codes can help reduce the impact of tokens being intercepted or exposed by clients. However, having tokens expire too quickly and requiring frequent reauthorization of access may impact the usability of the implementation. Consideration should be given to the requirements of particular use cases when determining reasonable lifetimes for tokens and codes.
* **OAuth Client Security** – The OAuth specification contains a good deal of security guidance on dealing with different types of clients. A web application hosted on a trusted partner’s server, for example, has very different security properties than a browser-based client. For each Client Delegation use case, an analysis is needed to support a number of decisions pertaining to clients:
  + What types of clients should be accommodated? OAuth divides clients into two broad categories: Confidential Clients, which can provide reasonable protection for client secrets (e.g., hosted web applications) and Public Clients, which cannot.
  + What degree of control over client registration is needed? This may range from explicitly enabling a small number of known, vetted clients to enabling dynamic registration of arbitrary clients. The decision to trust a given client is ultimately the Resource Owner’s; however, a malicious or incompetent client could expose authorization tokens as well as any data which it receives from the Resource Server.
  + Should clients be required to explicitly register their redirect Uniform Resource Identifiers (URIs)? Requiring clients to register a “white list” of allowed redirect URIs which is enforced by the authorization server can prevent some known attacks.
* **Token Security** – The OAuth standard does not specify a particular format for Access Tokens. As with other secrets, they must provide resistance against brute-force guessing attacks and be protected both during transmission and at rest. Using signed JWTs as the format for Access Tokens, as OpenID Connect does, could provide a useful means for Resource Servers to validate the authenticity of received tokens. The draft OAuth Token Introspection standard, which defines a new Authorization Endpoint server that can confirm the validity of submitted tokens and return additional token metadata, is another potentially useful option [16].
* **Protocol Flow Selection** – The OAuth specification provides guidance on selecting the appropriate protocol flow for different scenarios. Inappropriate use of the Implicit Flow, designed for browser-based clients that are directly controlled by the Resource Owner, could increase the risk of Authorization Tokens being intercepted by a compromised user agent or provided to an unauthorized server due to an unsafe redirect. The selection of the protocol flow to use for a particular OAuth use case calls for careful consideration and review of the security guidance in the specification.
* **Authorization Code and Access Token Replay Prevention** – The security guidance in the OAuth specification documents provides different options for preventing replay attacks, including the use of TLS to prevent interception, limiting the lifetime of tokens and codes, and using the *audience* parameter to specify the Resource Server(s) where Access Tokens are intended to be used.

## Identity Federation (VA as Relying Party) Pattern

### Description

In this pattern, the VA accepts authenticated identity information from an external OpenID Provider as a basis for granting a user access to VA resources. This pattern is very similar to VA’s current arrangement with DoD to enable user authentication with DoD Self-Service Logon (DS Logon) credentials. The difference is that DS Logon is based on the SAML standard, while this pattern is based on the use of OpenID Connect. An OpenID Connect-based federation capability could complement VA’s current SAML-based federation by enabling integration with additional providers that support OpenID Connect.

OAuth provides a framework for authorizing access to resources, but it does not convey any information about the authenticated user to the resource server. OpenID Connect builds an identity federation scheme on top of OAuth, enabling the Authorization Server to assert the authenticated identity of the user along with other user attributes and authentication context information.

In some cases, accepting identity and authentication information from a trusted external provider is preferable to issuing and maintaining local credentials for users. For example, if the users are employees of a partner organization, the partner organization will typically have more up-to-date information about the user’s employment status and other attributes that may be factors in an access control decision.

In this Identity Federation pattern, an external user authenticates to an external OP, which then provides an ID Token to a VA system. The ID Token conveys an assertion that the user identified in the token has been authenticated by the OP. The OP also provides the VA with an Access Token, which can be submitted to the OP’s UserInfo endpoint in order to obtain additional claims, or attributes, about the user. Figure 3 below illustrates this pattern.



Figure – Identity Federation (VA as RP) Pattern

### Specific Use Case Examples

Specific examples of the Identity Federation (VA as RP) pattern include:

* A VSO employee authenticates to a VA system through an OP operated by the VSO organization
* A specialist authenticates to a VA system through an OP operated by the external health care provider organization in order to view the EHR of a referred patient
* A pharmacy employee authenticates to a VA system through an OP operated by the pharmacy to upload veteran immunization records
* Veterans authenticate through the OP of their choice as an alternative to DS Logon

### Initiation

An external user (i.e., a user who does not possess VA credentials) initiates this use case by attempting to access a protected resource or application hosted at the VA. The resource server at the VA, the RP in this pattern, recognizes the user as having credentials at a trusted external OP of which it is a registered client, and initiates an Authentication Request to the OP.

### Use Case Actors

Table 2 lists the actors involved in the Identity Federation (VA as RP) pattern.

Table – Identity Federation (VA as RP) Use Case Actors

| Actor | Description |
| --- | --- |
| End User | A user of a VA application who does not have VA credentials, but does have credentials with the OP. The End User authorizes the Relying Party to access his/her identity information at the OP. |
| Relying Party | A VA-hosted application that is registered as an OpenID Connect client with the OP. |
| OpenID Provider | An OAuth 2.0 Authorization Server which responds to OpenID Connect Authentication Requests by authenticating the end user, requesting authorization for the RP to access its identity information, and issuing Identity Tokens and Access Tokens. The OP in this Pattern is hosted by an external party. |
| UserInfo Endpoint | An identity information service at the OP that accepts Access Tokens issued by the OP and returns claims about the End User. |

### Assumptions and Prerequisites

* The End User has a valid user account and credentials provisioned by the OP.
* The RP is a registered client of the OP, has been issued client credentials, and has the URLs for the OP’s OpenID Connect endpoints.
* The RP has evaluated the OP’s authentication and identity mechanisms and decided to trust claims issued by the OP to authenticate users.
* The RP and OP have agreed on the names and values of non-standard claims, if any, that may be returned from the UserInfo endpoint and used as inputs to access control decisions.

### Sequence of Events

Like OAuth, OpenID Connect supports multiple protocol flows. The following sequence is based on the Authorization Code flow. This flow sequence is depicted in Figure 4 and described below. Using the use case example involving a VSO employee authenticating to a VA application through a VSO-operated OP, the italicized text in the diagram indicates which role each of these parties would play in the protocol flow.

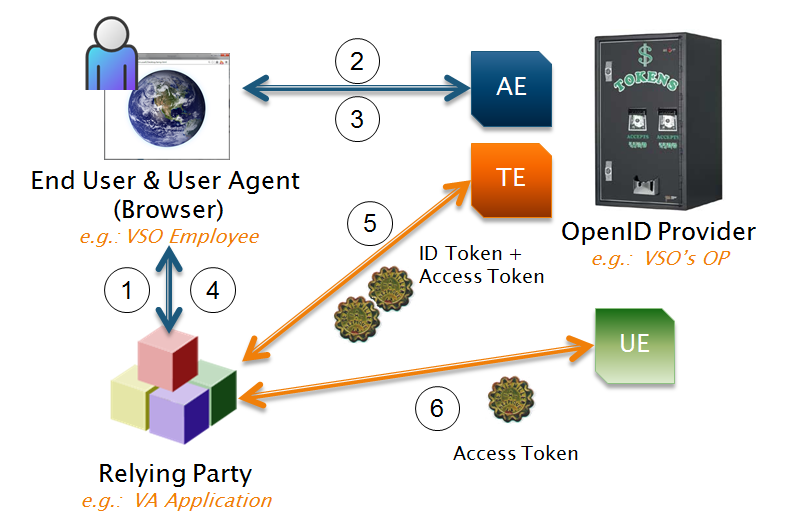


Figure – OpenID Connect Authorization Code Flow

1. An external (non-VA) End User attempts to access a VA-hosted application, the RP. The RP prompts the end-user to authenticate. The RP discovers that the user has an account with the external OP (e.g., by providing a “Sign in with your <OP Name> account” link which the user clicks).
2. The RP redirects the End User’s user agent to the OP’s Authorization Endpoint (AE) with an Authentication Request. In this pattern, the OP is operated by an organization external to the VA.
3. The OP prompts the End User to authenticate, and to authorize the VA RP to access his/her identity information.
4. If the End User authenticates successfully and authorizes access, the OP redirects the End User’s user agent back to the RP with an Authorization Code.
5. The RP submits its Authorization Code to the OP’s Token Endpoint (TE), and authenticates with its client credentials. If the OP successfully validates the Authorization Code and client authentication succeeds, the OP responds with both an ID Token and an Access Token.
6. The RP validates the ID Token and extracts the End User’s subject identifier (from the “sub” claim). Optionally, if the RP requires additional claims about the End User that are not included in the ID Token, the RP submits a UserInfo request to the OP’s UserInfo Endpoint (UE), along with the Access Token for authorization. The OP replies with a UserInfo Response, including additional claims about the End User.

Because OpenID Connect is based on OAuth, the protocol flow above could also be described purely in OAuth terms. In the OAuth context, the user’s identity information is the PR, the RP is the client, and the End User is the Resource Owner, who authorizes the client to access the identity information provided through the UserInfo endpoint.

### Post-conditions

* The RP possesses a signed set of claims from the OP asserting the user’s identity and optional additional information about the user’s authentication context or attributes of the user.
* The RP possesses an Access Token, which can be used to request additional claims from the user’s OP.
* The End User is authenticated to the RP application, without needing to establish or maintain additional credentials.

### Security Requirements

Because OpenID Connect is based on OAuth 2.0, this Identity Federation pattern implicitly inherits the same security considerations that apply to the Client Delegation pattern. Since the VA plays the role of OpenID Connect client in this instance, however, significant responsibility for the security of the solution rests with the external OP. Additional security requirements pertaining to the Identity Federation (VA as RP) pattern include:

* **Establishing Trust with OPs** – Before accepting identity and attribute assertions from an external OP, it is important to understand the OP’s policies and practices regarding account and credential provisioning, management of attributes, and general level of security hygiene and maintenance. Periodic re-evaluation would be required to ensure that the OP maintains appropriate practices for continued trust. As an alternative to direct engagement with OPs, the Federal Identity, Credential, and Access Management (FICAM) Trust Framework Provider (TFP) program reviews and approves Identity Services for use as identity providers by Federal agencies.
* **Preventing Token Replay Attacks** – The OpenID Connect specification suggests the use of a nonce value to prevent against the risk of unauthorized users intercepting and replaying valid tokens to gain access to the RP application. The RP is responsible for including the *nonce* parameter in the initial Authentication Request, and verifying that the same nonce value is returned in the *nonce* claim in the ID Token. By associating the *nonce* value with a unique session ID, the RP can ensure that the ID Token has not been substituted with one obtained through a different user’s session.
* **Determining External Users’ Authorizations** – Different approaches exist for determining the appropriate access permissions for federated users. If the OP reliably maintains standard attributes expressing pertinent information such as the user’s organizational role, they could be queried from the UserInfo endpoint and used in access control decisions. Another option is sponsorship, in which a VA user could assert the role or required authorities of an external user. It is critical to address the question of authorization as distinct from authentication. Trusting an OP to assert identities is an authentication matter, and does not imply that any and all identities asserted by the OP should be granted access to resources by the RP.
* **Authentication Context** – In many use cases, VA may need information about the user’s authentication context – for example, the type of credential(s) used to authenticate – in order to make an access control decision. The OpenID Connect specification defines two standard claims that enable the OP to communicate this information to the RP: “acr” (authentication context class reference), and “amr” (authentication methods reference). OpenID Connect also includes advanced options to enable the RP to require that the user authenticate to the OP with a specified authentication mechanism.
* **Responding to an OP Compromise** – Establishing a trust relationship with an OP should include defining security incident response procedures and designating security points of contact for the OP and RP organizations. In the event that an OP is compromised, the VA should immediately cease to accept ID Tokens from that OP until its organization can provide reasonable assurance that the compromise has been remediated. Re-establishment of trust will likely involve the issuance of new cryptographic keys, and VA would need to ensure that the compromised keys were no longer trusted. A review of audit logs would also be required to determine whether VA-hosted data was compromised.

## Identity Federation (VA as OpenID Provider) Pattern

### Description

This pattern is essentially the same as the previous Identity Federation pattern but with a reversal of roles. In this pattern, VA provides authenticated identity information to an external RP application, enabling VA users to access it with their existing credentials. While the previous pattern required the VA to establish a trust relationship with an OP to provide authenticated user information, in this case the VA would trust the external party with access to information about its own users. The benefit to the VA’s users would be access to external applications without the need to establish new credentials.

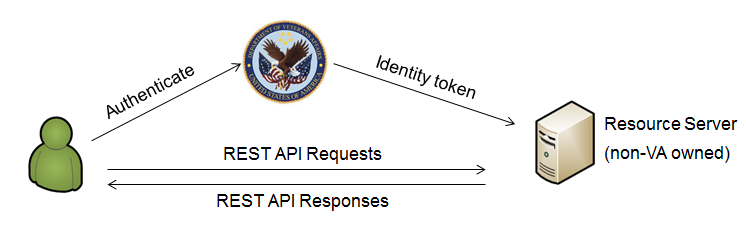


Figure – Identity Federation (VA as OP) Pattern

### Specific Use Case Examples

Specific examples of the Client Delegation pattern include:

* A VA physician accesses an external health care provider’s web portal to access records of a veteran’s referred consultation with a specialist
* A VA cybersecurity analyst accesses an external Poll service to obtain cyber threat information using a service such as Trusted Automated Exchange of Indicator Information (TAXII) [16]

### Initiation

A VA user initiates this use case by attempting to access an external protected resource or application. The external resource server, the RP in this pattern, recognizes the user as having credentials at the VA’s OpenID Connect OP, of which it is a registered client, and initiates an Authentication Request to the OP.

### Use Case Actors

Table 3 lists the actors involved in the Identity Federation (VA as OP) pattern.

Table – Identity Federation (VA as OP) Use Case Actors

| Actor | Description |
| --- | --- |
| End User | A VA user attempting to access the external RP application. The End User authorizes the Relying Party to access its identity information at the VA OP. |
| Relying Party | An externally-hosted application that is registered as an OpenID Connect client with the VA’s OP. |
| OpenID Provider | A VA-hosted OAuth 2.0 Authorization Server which responds to OpenID Connect Authentication Requests by authenticating the end user, requesting authorization for the RP to access its identity information, and issuing Identity Tokens and Access Tokens. |
| UserInfo Endpoint | An identity information service at the OP that accepts Access Tokens issued by the OP and returns claims about the End User. |

### Assumptions and Prerequisites

* The End User has a valid user account and credentials provisioned by VA.
* The Relying Party is a registered client of VA’s OP, has been issued client credentials, and has the URLs for the OP’s OpenID Connect endpoints.
* The RP has evaluated the OP’s authentication and identity mechanisms and decided to trust claims issued by the OP to authenticate users.
* The RP and OP have agreed on the names and values of non-standard claims, if any, that may be returned from the UserInfo endpoint and used as inputs to access control decisions.

### General Sequence of Events

Like OAuth, OpenID Connect supports multiple protocol flows. The following sequence is based on the Authorization Code flow. The protocol flow is the same as the one presented in section 4.2.6, apart from the reversal of roles between the VA and the external organization. This flow sequence is depicted in Figure 6 and described below. Using the use case example involving a VA user authenticating to an external health care provider’s web portal through a VA-operated OP, the italicized text in the diagram indicates which role each of these parties would play in the protocol flow.

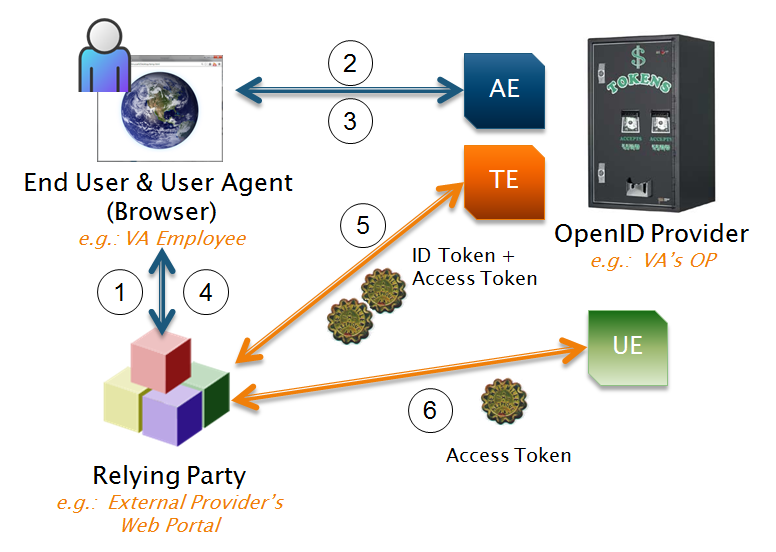


Figure – OpenID Connect Authorization Code Flow

1. The End User attempts to access the external RP; the RP prompts the end-user to authenticate. The RP discovers that the user has an account with the VA OP (e.g., by providing a “Sign in with your VA account” link which the user clicks).
2. The RP redirects the End User’s user agent to the VA OP with an Authentication Request.
3. The OP prompts the End User to authenticate, and to authorize the RP to access his/her identity information.
4. If the End User authenticates successfully and authorizes access, the OP redirects the End User’s user agent back to the RP with an Authorization Code.
5. The RP submits its Authorization Code to the OP, and authenticates with its client credentials. If the OP successfully validates the Authorization Code and client authentication succeeds, the OP responds with both an ID Token and an Access Token.
6. The RP validates the ID Token and extracts the End User’s subject identifier (from the “sub” claim). Optionally, if the RP requires additional claims about the End User that are not included in the ID Token, the RP submits a UserInfo request to the OP’s UserInfo endpoint, along with the Access Token for authorization. The OP replies with a UserInfo Response, including additional claims about the End User.

### Post-conditions

* The RP possesses a signed set of claims for the OP asserting the user’s identity and optional additional information about the user’s authentication context or other attributes of the user.
* The RP possesses an Access Token, which can be used to request additional claims from the user’s OP.
* The End User is authenticated to the RP application, without needing to establish or maintain additional credentials.

### Security Requirements

Because OpenID Connect is based on OAuth 2.0, this Identity Federation pattern implicitly inherits the same security considerations that apply to the Client Delegation pattern. This Pattern also shares many security concerns with the VA as RP Federation pattern, though the distribution of security responsibilities and risks between VA and RP are substantially different here.

* **Establishing Trust with RPs** – Integration with an RP does not entail the same degree of trust as accepting identity assertions from an OP, but the OP does expose information about its internal user accounts through the issuance of ID Tokens and UserInfo responses. The requirement for End User authentication and authorization of the RP’s access prevents the RP from extracting identity information without the user’s involvement. However, the RP may abuse the identity information of authenticated users or expose it to unauthorized third parties through malicious intent or poor security practices.
* **Obtaining User Authorization** – The OP is responsible for providing the End User with information to make an informed decision on whether to allow the RP to access identity information. The user interface must effectively convey information about the scope of the request and the identity of the requesting site to enable these decisions. Likewise, the OP should provide an interface to allow the user to view, manage, and revoke existing authorizations to RP applications.

## Potential Future Pattern – User-Managed Access

While the OAuth 2.0 Framework is now well-established, this pattern is based on the draft UMA standard [17], which is still evolving. A few real-world UMA deployments exist and some software implementations are available. Also, UMA may see wider adoption over the next few years.

UMA is an OAuth 2.0-based access management protocol. To the standard OAuth set of players including a Resource Owner, Client, Authorization Server, and Protected Resource, UMA adds a Requesting Party, enabling the resource owner to create policies authorizing specific individuals to access resources through a given client. The UMA Architecture is depicted in Figure 7.

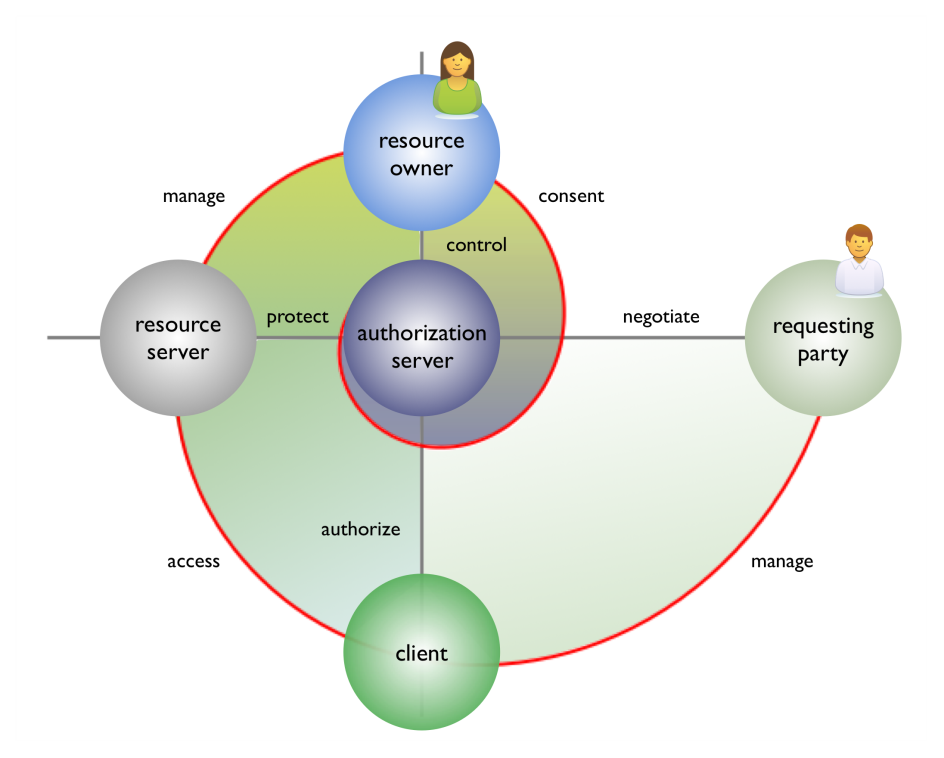


Figure - UMA Architecture [[2]](#footnote-2)

UMA introduces the ability for the Resource Owner to create policies that the authorization server can use to make automated authorization decisions based on specified criteria, rather than requiring the owner to explicitly approve each authorization request. Examples include requiring the requesting party to have certain identity attributes, to have a stated privacy policy meeting requirements specified by the owner, or to agree to non-disclosure terms. Of course, some of these criteria would require self-enforcement by the requesting party, but they could be backed up by contractual agreements among the parties to a given exchange.

As a centralized authorization service that could protect many different Resource Servers, UMA’s Authorization Server bears some resemblance to a Policy Decision Point (PDP) as defined in the Extensible Access Control Markup Language (XACML) specification. As some of the large web properties such as Google and Facebook have expanded their roles to become identity providers to other sites, it is conceivable that they could also operate internet-scale authorization services. However, even for enterprises, while many applications have readily delegated user authentication and account management responsibilities to external providers, the adoption of centralized authorization services has not been widespread.

In the health IT domain, UMA has great potential for advancing patient privacy controls in EHR systems. Some potential VA health use cases for UMA include:

* A veteran delegates a specialist, to whom he/she has been referred at a third-party health care provider organization, access to portions of his/her EHR
* A veteran delegates access and management authority to his/her EHR to a designated family member or VSO
* Veterans are provided with a privacy “dashboard,” enabling them to define a set of policies governing access to their medical records under different conditions, including various types of research, predictive diagnostics, and genomic analysis

UMA is a complex standard, incorporating three separate OAuth client delegation flows in the core protocol. The protocol flow is depicted in Figure 8.

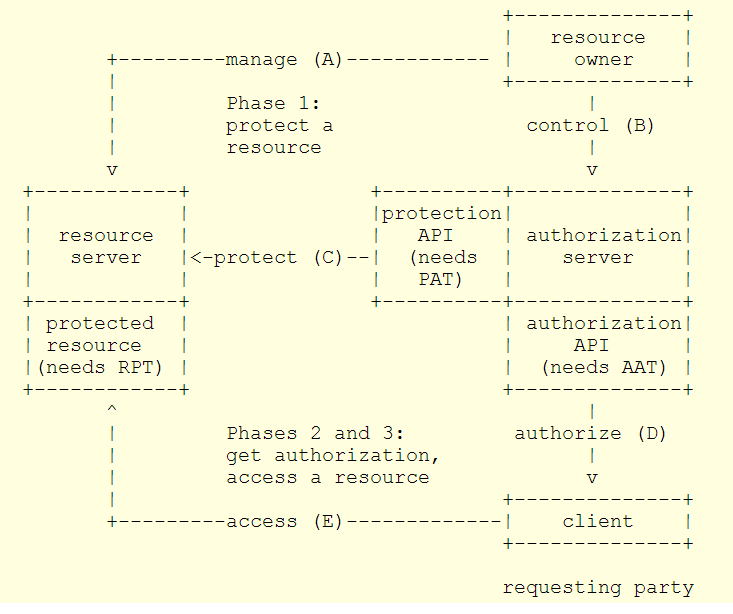


Figure - UMA Core Protocol Flow[[3]](#footnote-3)

Phase One involves the Resource Owner, the Resource Server, and the Authorization Server. The Resource Owner registers the Resource Server to interact with the Authorization Server’s protection API. This is a typical OAuth client delegation flow, in which the Authorization Server’s protection API is the Protected Resource, and the Resource Server is the client. The Access Token issued to the Resource Server is referred to as the Protection API Token (PAT).

Following Phase One, the Resource Owner defines policies to control access to the protected resources on the Resource Server; the policy mechanisms are outside the scope of the UMA standard. They could involve authorizing particular named individuals or specifying criteria that the Requesting Party must meet in order to be authorized.

In Phase Two, a Requesting Party uses a Client to attempt to access a protected resource on the Resource Server, but does not have the required authorization token. The Resource Server redirects the Requesting Party to the Authorization Server. At this point there is another OAuth client delegation flow, in which the Requesting Party (acting as Resource Owner) authorizes the Client to use the Authorization Server’s Authorization API. This results in the issuance of an Authorization API Token (AAT) to the client.

The Authorization Server issues the Client a Requesting Party Token (RPT), which the Client then presents to the Resource Server along with a request to access the protected resource. At this point, the Resource Server may require the Requesting Party to provide additional identity claims in order to satisfy the UMA policies applied to the resource (e.g., to prove that the Requesting Party has the required attributes for access authorization).

Finally, if the Requesting Party can provide the claims required to satisfy the policy, Phase 3 follows, in which the Requesting Party is granted access to the resource.

It is difficult to analyze the security of the UMA protocol without implementation specifics such as the distribution of its components among different organizations, and their relationships with each other. While complexity in protocols is generally bad for security, UMA’s three-phase protocol flow is composed of three distinct OAuth flows that could be analyzed independently.

The adoption of UMA for VA use cases may not be practical in the near term, but it is presented here as an example of the potential future use cases that could be enabled by RESTful APIs, OAuth, and related standards.

# Next Steps

This document has presented a set of REST security patterns and candidate VA use cases as a foundation for developing a Secure RESTful Interfaces Profile. The project team intends to collaborate broadly across VA stakeholders throughout the development of this guidance, and it is envisioned that additional use cases and requirements may be introduced through this collaboration. As stated in Section 2, the initial set of use cases included in this document is a representative sample and not exhaustive.

RESTful interfaces have the potential to serve a number of VA use cases, and the security standards discussed in this document have particularly useful applications for providing patients with greater control over their medical data. The Secure RESTful Interfaces Profile produced through this project will help VA to adopt the REST approach in a way that provides the needed protection of data and applications.

# List of Acronyms

| Acronym | Definition |
| --- | --- |
| AAT | Authorization API Token |
| AE | Authorization Endpoint |
| API | Application Programming Interface |
| ASD | Architecture, Strategy, and Design |
| BB+ | BlueButton+ |
| CA | Certificate Authority |
| DoD | Department of Defense |
| DS Logon | DoD Self-service Logon |
| EHR | Electronic Health Record |
| FHIR | Fast Healthcare Interoperability Resources |
| FICAM | Federal Identity, Credential, and Access Management |
| HL7 | Health Level Seven |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | HTTP Secure |
| IdP | Identity Provider |
| IETF | Internet Engineering Task Force |
| IT | Information Technology |
| JOSE | JSON Object Signing and Encryption |
| JSON | JavaScript Object Notation |
| JWA | JSON Web Algorithms |
| JWE | JSON Web Encryption |
| JWK | JSON Web Key |
| JWS | JSON Web Signature |
| JWT | JSON Web Token |
| MAC | Message Authentication Code |
| MITM | Man-in-the-Middle |
| OIT | Office of Information & Technology |
| ONC | Office of the National Coordinator for Health IT |
| OP | OpenID Provider |
| PAT | Protection API Token |
| PDP | Policy Decision Point |
| PGD | Patient-Generated Data |
| PHI | Protected Health Information |
| REST | Representational State Transfer |
| RFC | Request for Comments |
| RHEx | RESTful Health Exchange |
| RP | Relying Party |
| RPT | Requesting Party Token |
| SAML | Security Assertion Markup Language |
| SOA | Service-Oriented Architecture |
| TAXII | Trusted Automated Exchange of Indicator Information |
| TE | Token Endpoint |
| TFP | Trust Framework Provider |
| TLS | Transport-Layer Security |
| UMA | User-Managed Access |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| VA | Veterans Affairs |
| VistA | Veterans Health Information Systems & Technology Architecture |
| VSO | Veterans Service Organization |
| XACML | Extensible Access Control Markup Language |
| XML | Extensible Markup Language |

# References

|  |  |
| --- | --- |
| [1] | Department of Veterans Affairs, "FY 2014-2020 Strategic Plan," March 2014. [Online]. Available: http://www.va.gov/op3/docs/StrategicPlanning/VA2014-2020strategicPlan.PDF. [Accessed May 2014]. |
| [2] | R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. [Accessed 2014 May]. |
| [3] | A. DuVander, "New Job Requirement: Building RESTful APIs," ProgrammableWeb, 9 June 2010. [Online]. Available: http://www.programmableweb.com/news/new-job-requirement-experience-building-restful-apis/2010/06/09. [Accessed May 2014]. |
| [4] | Health Level Seven International, "FHIR Specification," 2014. [Online]. Available: http://www.hl7.org/implement/standards/fhir/. [Accessed May 2014]. |
| [5] | HIT Standards Committee NwHIN Power Team, "Transport Standards for Consumer Exchanges: Preliminary," 19 June 2013. [Online]. Available: http://www.healthit.gov/facas/sites/faca/files/2013jun19\_hitsc\_nwhin\_0.pdf. [Accessed May 2014]. |
| [6] | S&I Framework, "RHEx - Powering Secure, Web Base Health Data Exchange," [Online]. Available: http://wiki.siframework.org/RHEx. [Accessed May 2014]. |
| [7] | S&I Framework, "BlueButton Plus Initiative," [Online]. Available: http://wiki.siframework.org/BlueButton+Plus+Initiative. [Accessed 2014 May]. |
| [8] | T. Dierks and E. Rescorla, "The Tansport Layer Security (TLS) Protocol Version 1.2," August 2008. [Online]. Available: http://tools.ietf.org/html/rfc5246. [Accessed May 2014]. |
| [9] | D. Hardt, "The OAuth 2.0 Authorization Framework," October 2012. [Online]. Available: http://tools.ietf.org/html/rfc6749. [Accessed May 2014]. |
| [10] | M. Jones, J. Bradley and N. Sakimura, "JSON Web Token (JWT)," 30 April 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-20. [Accessed May 2014]. |
| [11] | M. Jones, "JSON Web Signature (JWS)," 30 April 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-26. [Accessed May 2014]. |
| [12] | M. Jones and J. Hildebrand, "JSON Web Encryption (JWE)," 30 April 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-26. [Accessed May 2014]. |
| [13] | M. Jones, "JSON Web Key (JWK)," 30 April 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-jose-json-web-key-26. [Accessed May 2014]. |
| [14] | M. Jones, "JSON Web Algorithms (JWA)," 15 September 2013. [Online]. Available: https://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-16. [Accessed May 2014]. |
| [15] | N. e. a. Sakimura, "OpenID Connect Core 1.0," 25 February 2014. [Online]. Available: http://openid.net/specs/openid-connect-core-1\_0.html. [Accessed May 2014]. |
| [16] | T. Lodderstedt, M. McGloin and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations," January 2013. [Online]. Available: http://tools.ietf.org/html/rfc6819#page-46. [Accessed May 2014]. |
| [17] | W. Jing, "Covert Redirect Vulnerability Related to OAuth 2.0 and OpenID," May 2014. [Online]. Available: http://tetraph.com/covert\_redirect/oauth2\_openid\_covert\_redirect.html. [Accessed May 2014]. |
| [18] | J. Richer, "OAuth Token Introspection," 1 May 2013. [Online]. Available: http://tools.ietf.org/html/draft-richer-oauth-introspection-04. [Accessed May 2014]. |
| [19] | The MITRE Corporation, "TAXII Version 1.1," [Online]. Available: http://taxii.mitre.org/specifications/version1.1/. [Accessed May 2014]. |
| [20] | T. Hardjono, "User-Managed Access (UMA) Profile of OAuth 2.0," 4 March 2014. [Online]. Available: http://docs.kantarainitiative.org/uma/draft-uma-core.html. [Accessed May 2014]. |
| [21] | L. Richardson, M. Amundsen and S. Ruby, RESTful Web APIs, Cambridge: O'Reilly, 2013. |

1. SOAP originally expanded as Simple Object Access Protocol, but this acronym was dropped with the standard

   version defined in the W3C Recommendation for SOAP 1.2 [↑](#footnote-ref-1)
2. Diagram source: https://kantarainitiative.org/confluence/display/uma/Home [↑](#footnote-ref-2)
3. Diagram source: [17] [↑](#footnote-ref-3)