2 COLOR STATE SEAL (blk

Version 1.0

February 11, 2020

by: Dina Heidar

State of Louisiana



IdentityServer4 Technical Design Document

TABLE OF CONTENTS

[Revision History 3](#_Toc32398974)

[Introduction 4](#_Toc32398975)

[Purpose 4](#_Toc32398976)

[Audience 4](#_Toc32398977)

[Acronyms, Abbreviations, Terms and Definitions 4](#_Toc32398978)

[Design Overview 5](#_Toc32398979)

[Approach 5](#_Toc32398980)

[Security Requirements 5](#_Toc32398981)

[Architectural Goals and Constraints 5](#_Toc32398982)

[Technologies Used 5](#_Toc32398983)

[What is IdentityServer? 6](#_Toc32398984)

[OpenID Connect and OAuth 2.0 Protocols Flows 6](#_Toc32398985)

[Which OpenID Connect/OAuth 2.0 Flow is the right one? 6](#_Toc32398986)

[Tokens 7](#_Toc32398987)

[Identity Token 7](#_Toc32398988)

[Access Token 7](#_Toc32398989)

[Response Types by Flow 7](#_Toc32398990)

[Response Types 7](#_Toc32398991)

[Un-authenticated user using Authorization Code Flow and Proof Key for Code Exchnage (PKCE) 10](#_Toc32398992)

[UN-AUTHENTICATED User USING HyBrid Flow and EncryPTION 13](#_Toc32398993)

[Topology Diagram 14](#_Toc32398994)

[Features 15](#_Toc32398995)

[References 16](#_Toc32398996)

[Appendix A 17](#_Toc32398997)

**TABLE OF FIGURES**

[Figure 1: Authorization Code Flow and PKCE 12](file:///C:\Users\dinah\source\repos\IdentityServer\documents\IdentityServer4%20Technical%20Design%20Document.docx#_Toc32393505)

[Figure 2: Existing User 12](file:///C:\Users\dinah\source\repos\IdentityServer\documents\IdentityServer4%20Technical%20Design%20Document.docx#_Toc32393506)

[Figure 3: System Integration 13](#_Toc32393507)

## Revision History

| Version | Date | Description | Editor |
| --- | --- | --- | --- |
| **1.0** | 2/11/2020 | Initial Draft | Dina Heidar |

## Introduction

### Purpose

The purpose of this project is show how IdentityServer4 can:

1. Authenticate users via the clients to external non oidc/oauth IdentityProviders (e.g. ADFS, CA IAM…. etc.) obtain in return an *access\_token* to communicate with apis(e.g. SAML token to Bearer)
2. Eliminate the need for clients to create jwts for apis thus reducing update certificate maintenance issues when they expire (for both the clients and apis).
3. Create and verify jwts between clients and apis.
4. Obtain *access\_tokens* and *id\_tokens* without using *client\_secrets.*.
5. Obtain *access\_tokens* that contain the necessary client claims when actions are performed without users.
6. Have *access\_tokens* contain the necessary user claims when api actions are performed on behalf of a user.
7. Ensure that *id\_tokens* are either encrypted (JWE) over the browser when using the Hybrid flow <https://www.scottbrady91.com/Identity-Server/Encrypting-Identity-Tokens-in-IdentityServer4>.
8. Ensure *id\_tokens* not exposed at all by using the PKCE and Authorization Code flow <https://www.scottbrady91.com/OpenID-Connect/ASPNET-Core-using-Proof-Key-for-Code-Exchange-PKCE>. (.NetCore >=3.0)
9. Add/remove/verify roles, claims and locations for users. (using identity and possibly AdminUI)
10. Utilize custom delegation grants. <https://www.scottbrady91.com/OAuth/Delegation-Patterns-for-OAuth-20>

### Audience

The intended audience for this document is State of Louisiana agency business technical architects and developers.

### Acronyms, Abbreviations, Terms and Definitions

Please refer to Appendix A for the list of all acronyms and abbreviations.

## Design Overview

### Approach

### Security Requirements

### Architectural Goals and Constraints

## Technologies Used

The target platform can be either Windows or Linux. The framework used is .NET Core 3.1 and the development environment is Microsoft Visual Studio.

## What is IdentityServer?

IdentityServer is an OpenID Connect provider - it implements the OpenID Connect and OAuth 2.0 protocols. The protocol flows are defined by setting a grant type(s) to clients. These grant types specify how a client can interact with the token service.

IdentityServer has a number of jobs and features - including:

* protect your resources
* authenticate users using a local account store or via an external identity provider
* provide session management and single sign-on
* manage and authenticate clients
* issue identity and access tokens to clients
* validate tokens

## OpenID Connect and OAuth 2.0 Protocols Flows

OAuth 2.0 is a security standard where you give one application permission to access your data in another application. The steps to grant permission, or consent, are often referred to as authorization or even delegated authorization. You authorize one application to access your data, or use features in another application on your behalf, without giving them your password

OAuth 2.0 is designed only for authorization, for granting access to data and features from one application to another. OpenID Connect (OIDC) is a thin layer that sits on top of OAuth 2.0 that adds login and profile information about the person who is logged in. ... The OpenID Connect flow looks the same as OAuth.

The OpenID Connect flow looks the same as OAuth. The only differences are, in the initial request, a specific scope of *openid* is used, and in the final exchange the Client receives both an Access Token and an ID Token.

With OIDC, there’s also a standard way the Client can request additional identity information such as their email address, using the Access Token.

More info at <https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc>

## Which OpenID Connect/OAuth 2.0 Flow is the right one?

**Machine to Machine Communication**: Client Credentials Flow

**Browser-based Applications (SPA)**: OAuth2 working group now recommends Authorization Code with PKCE vs previously used Implicit.

**Native Applications**:  Authorization Code with PKCE (with no secret since it is useless) <https://www.scottbrady91.com/OpenID-Connect/ASPNET-Core-using-Proof-Key-for-Code-Exchange-PKCE> OR Hybrid with encrypted JWT <https://www.scottbrady91.com/Identity-Server/Encrypting-Identity-Tokens-in-IdentityServer4>.

.

## Types of Tokens

### Identity Token

An *id\_token* represents the outcome of an authentication process. It contains at a bare minimum an identifier for the user (called the sub aka subject claim) and information about how and when the user authenticated. It can contain additional identity data. Identity data represent user claims and more.

### Access Token

An *access\_token* allows access to an API resource. Clients request access tokens and forward them to the API. Access tokens contain information about the client and the user (if present). APIs use that information to authorize access to their data.

## Response Types by Flow

|  |  |
| --- | --- |
| Flow | Response Types |
| Authorization Code | code |
| Implicit | id\_token |
| Implicit | id\_token token |
| Hybrid | code id\_token |
| Hybrid | code token |
| Hybrid | code id\_token token |

## Response Types

<https://medium.com/@darutk/diagrams-of-all-the-openid-connect-flows-6968e3990660>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Response Type | Scope | Authorization Code | Access Token | ID Token |
| code | openid | **🗸** | **🗸** | **🗸** |
|  | **🗸** | **🗸** |  |
| token | openid |  | **🗸** |  |
|  |  | **🗸** |  |
| id\_token |  |  |  | **🗸** |
| id\_token token |  |  | **🗸** | **🗸** |
| code id\_token |  | **🗸** | **🗸** | **🗸** |
| code token | openid | **🗸** | **🗸** | **🗸** |
|  | **🗸** | **🗸** |  |
| code id\_token token |  | **🗸** | **🗸** | **🗸** |

### Un-authenticated user using Authorization Code Flow and Proof Key for Code Exchnage (PKCE)

During authentication, mobile/native applications can use the OAuth 2.0 Authorization Code Flow, but they require additional security because they:

Cannot securely store a Client Secret. Decompiling the app will reveal the Client Secret. The Client Secret is bound to the app and is the same for all users and devices.

May make use of a custom URL scheme to capture redirects (e.g., MyApp://) potentially allowing malicious applications to receive an Authorization Code from your Authorization Server.

To mitigate this, OAuth 2.0 provides a version of the Authorization Code Flow which makes use of a Proof Key for Code Exchange (PKCE) (defined in [OAuth 2.0 RFC 7636](https://tools.ietf.org/html/rfc7636)).

The PKCE-enhanced Authorization Code Flow introduces a secret created by the calling application that can be verified by the authorization server; this secret is called the Code Verifier. Additionally, the calling app creates a transform value of the Code Verifier called the Code Challenge and sends this value over HTTPS to retrieve an Authorization Code. This way, a malicious attacker can only intercept the Authorization Code, and they cannot exchange it for a token without the Code Verifier

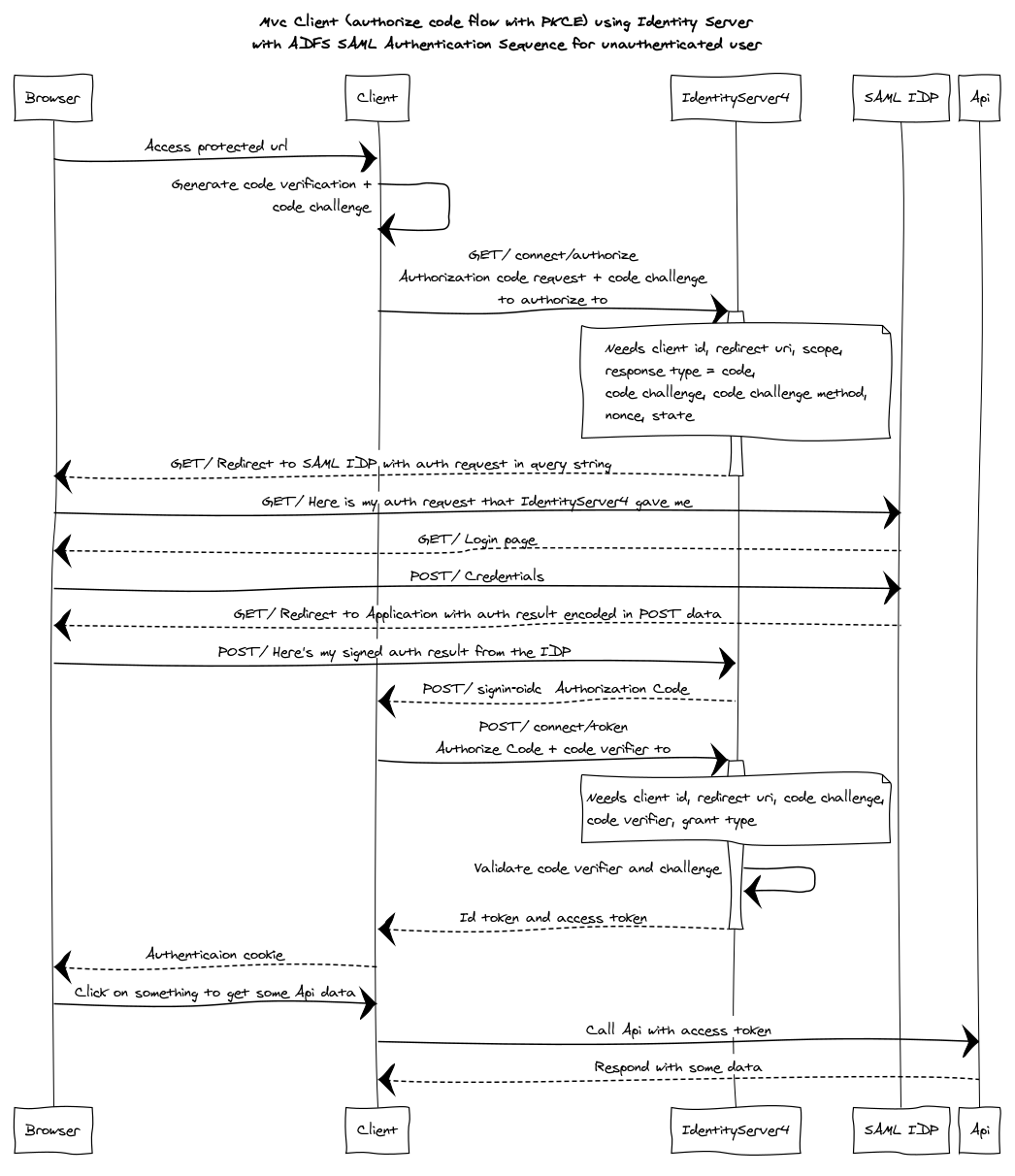


Figure 1: Authorization Code Flow and PKCE

### UN-AUTHENTICATED User USING HyBrid Flow and EncryPTION

Figure 2: Hybrid Flow and Encryption

Figure 2: Existing User

## Topology Diagram

Figure 3: System Integration

## Features

## References

## Appendix A

C

CEK: Content Encryption Key, 7

E

EA: Enterprise Architecture, 6

ESB: Enterprise Service Bus, 6

I

IAM: Identity Access Management, 6

IDP: Identity Provider, 6

J

JWE: JSON Web Encryption, 6

JWS: JSON Web Signature, 6

JWT: JSON Web Token, 6

O

OTS: Office of Technology Services, 7

S

SAML: Security Assertion Markup Language, 6

SOAP: Simple Object Access Protocol, 6

SSO: Single Sign-On, 6

*This page intentionally left blank*