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](#_Toc32332483)

[Introduction 4](#_Toc32332484)

[Purpose 4](#_Toc32332485)

[Audience 4](#_Toc32332486)

[Acronyms, Abbreviations, Terms and Definitions 4](#_Toc32332487)

[Design Overview 5](#_Toc32332488)

[Approach 5](#_Toc32332489)

[Security Requirements 5](#_Toc32332490)

[Architectural Goals and Constraints 5](#_Toc32332491)

[Technologies Used 5](#_Toc32332492)

[Flows 5](#_Toc32332493)

[Un-authenticated user using Authorization Code Flow and Proof Key for Code Exchnage (PKCE) 5](#_Toc32332494)

[UN-AUTHENTICATED User USING HyBrid Flow and EncryPTION 7](#_Toc32332495)

[Topology Diagram 8](#_Toc32332496)

[Features 9](#_Toc32332497)

[References 10](#_Toc32332498)

[Appendix A 11](#_Toc32332499)

**TABLE OF FIGURES**

[Figure 1: New User 9](file:///C:\Users\dinah\source\MyLa\Documentation\MyLa.Gov%20Design%20Document.docx#_Toc11318075)

[Figure 2: Existing User 10](file:///C:\Users\dinah\source\MyLa\Documentation\MyLa.Gov%20Design%20Document.docx#_Toc11318076)

[Figure 3: System Integration 11](#_Toc11318077)

## 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.

## Flows

### 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

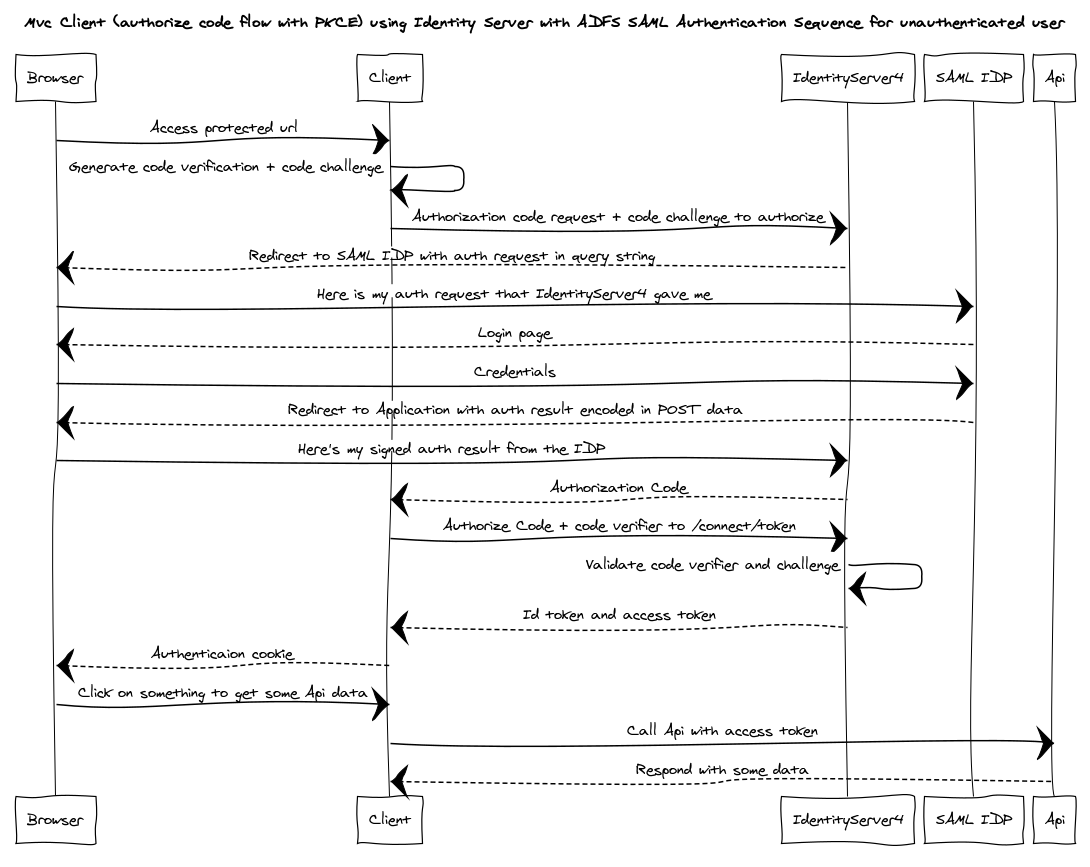


Figure 1: Authorization Code Flow and PKCE

### 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*