

Configuration Manual

MSc Research Project
MSc in Cybersecurity

Ketki Shekhar Jakatdar
Student ID: x22152229

School of Computing
National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name:Ketki Shekhar Jakatdar.....

Student ID:x22152229.....

Programme:MSc in Cybersecurity..... **Year:** 2023-2024

Module: ...MSc Research Practicum Part 2.....

Lecturer: ...Prof. Vikas Sahni.....

Submission Due Date: ...12/08/2024.....

Project Title: Leveraging X.509 Certificates and OAuth for optimized use of DIDs and VCs in Constrained IoT Devices.....

Word Count:2273..... **Page Count:**32.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ...Ketki Shekhar Jakatdar.....

Date: ...12/08/2024.....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ketki Shekhar Jakatdar
X22152229

1 Introduction

This document provides detailed specifications and guidelines for the tools, software, and hardware used in the implementation of the research project. The research aimed to innovate a secure and efficient solution that leverages Decentralized Identifiers (DID) and Verifiable Credentials (VC) along with ACE-OAuth and X509 certificates, to authenticate and authorize users trying to access constrained IoT devices. The purpose of this manual is to enable others to accurately recreate the project by providing all necessary links, commands, and step-by-step instructions along with screenshots. All the necessary URLs to README pages or to download the tools, are added as footnotes in the document.

2 System Configuration

Here, detailed specifications of the system used for the implementation is given. *Table 1* outlines the hardware and Operating System (OS) configuration, while *Table 2* lists the software tools and their respective versions installed on the system.

Table 1: Hardware and OS specification

Specification	Details
Processor	Intel Core i5-1335US
RAM	8GB
Storage	512GB SSD
Operating System (OS)	Microsoft Windows 11 Home

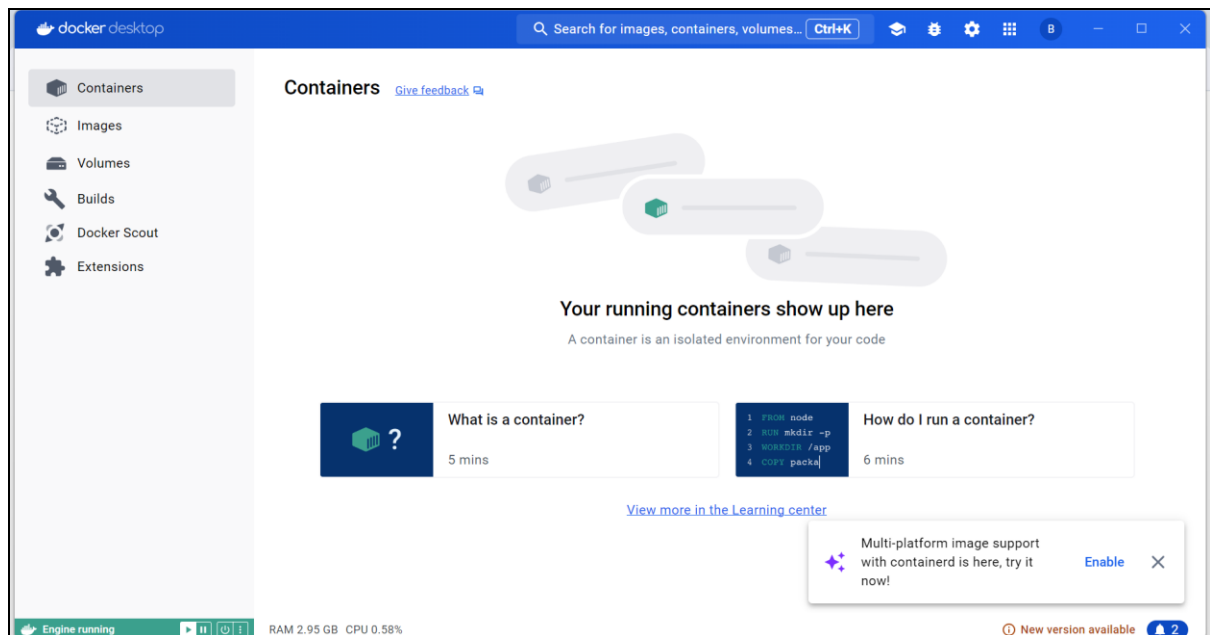
Table 2: Tools and their versions installed

Tool	Version
Python	3.12.5
PHP	8.3.9
Composer	2.7.7
Apache	2.4.62 (Win64)
Docker Desktop	26.1.4
VS Code	1.92.0 x64
Git	2.45.2.windows.1
MySQL Workbench	8.0.38 (64 bits) community
Postman	v11.7.0
Burp Suite	v2024.5.5
Wireshark	4.2.6
OpenSSL	3.3.1 4
Node.js	v20.16.0
Express/NPM	10.8.1
TypeScript	5.5.4
MySQL Workbench	8.0.38

3 Installation and Setup

3.1 Hyperledger Aries-Cloudagent-Python

- First, download and install Python 3.12.5 from [HERE](#)¹.
- Download and install Git 2.45.2 Standalone Installer for Windows from [HERE](#)².
- Download and install Docker Desktop from [HERE](#)³. Upon successful installation, the app will open home screen like below:



- To run the Hyperledger Aries-Cloudagent-Python project on docker, we need a VON network which is basically a Hyperledger Indy public ledger sandbox. Follow the *“Building and Starting”* steps from [HERE](#)⁴.

¹ <https://www.python.org/downloads/>

² <https://git-scm.com/download/win>

³ <https://www.docker.com/products/docker-desktop/>

⁴ <https://github.com/bcgov/von-network/blob/main/docs/UsingVONNetwork.md#building-and-starting>

```

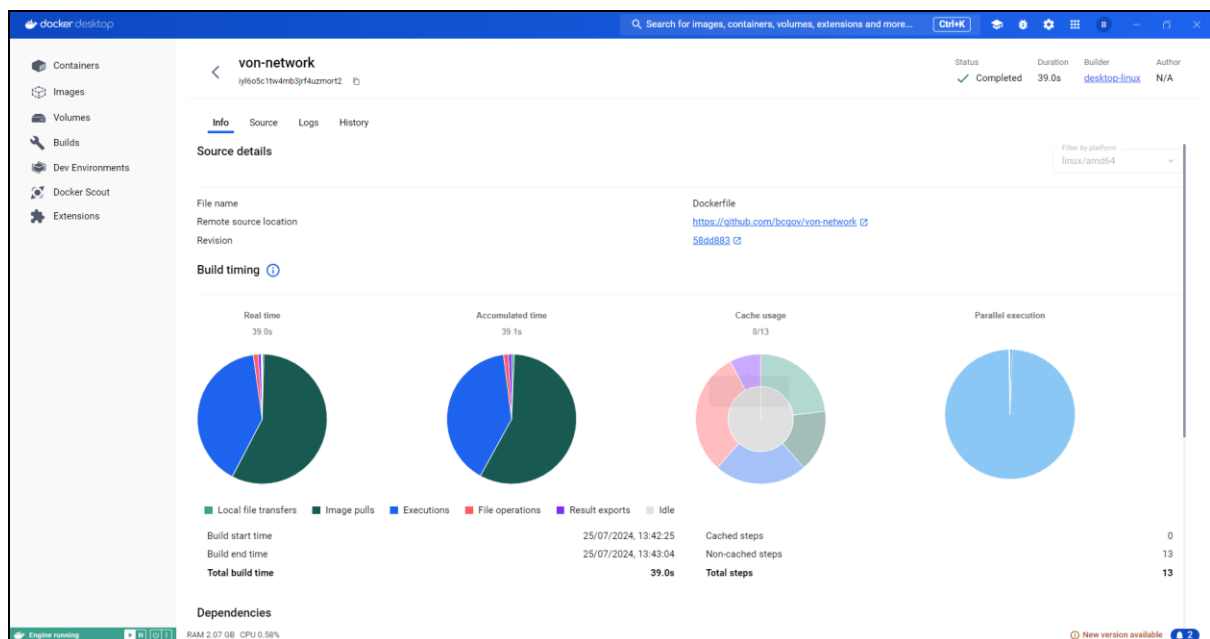
MINGW64~/Users/ketkijaka x + v
ketki.jakatdar@Ketki MINGW64 ~
$ git clone https://github.com/bcgov/von-network
Cloning into 'von-network'...
remote: Enumerating objects: 2221, done.
remote: Counting objects: 100% (2221/2221), done.
remote: Compressing objects: 100% (827/827), done.
Receiving objects: 100% (2221/2221), 497.03 KiB | 10.35 MiB/s, done.
remote: Total 2221 (delta 1352), reused 2108 (delta 1338), pack-reused 0
Resolving deltas: 100% (1352/1352), done.

ketki.jakatdar@Ketki MINGW64 ~
$ cd von-network

ketki.jakatdar@Ketki MINGW64 ~/von-network (main)
$ ./manage build
Creating ./tmp ...
[+] Building 39.1s (14/14) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 798B
=> [internal] load metadata for docker.io/bcgovimages/von-image:node-1.12-6
=> [auth] bcgovimages/von-image:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 158B
=> [1/8] FROM docker.io/bcgovimages/von-image:node-1.12-6@sha256:55f1d0ed54f1a79a2e041c5a32d9555f037d0eeb28654a33ea99153c36a
=> => resolve docker.io/bcgovimages/von-image:node-1.12-6@sha256:55f1d0ed54f1a79a2e041c5a32d9555f037d0eeb28654a33ea99153c36a
=> => sha256:0add632ed1e554a9f6ab69b6b816584272999d7befb0b8d11df6f1f66bab3aeb 51.10MB / 51.10MB
=> => sha256:55f1d0ed54f1a79a2e041c5a32d9555f037d0eeb28654a33ea99153c36aa05be 2.83kB / 2.83kB
=> => sha256:0da436103ea8426c0a9e26744f3e155b673acf4d8751233383ec81f675da7a17 4.52kB / 4.52kB
=> => sha256:7c6993a9bd27d12099c49827eb84350d131bcae05278c67a4a97b2013d70b588 8.77kB / 8.77kB
=> => sha256:22c5ef60a68e3fa0bd8df52f48470e836e823434a3bead43e35e87098a0b01fa 26.71MB / 26.71MB
=> => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B
=> => sha256:9a0d526fdbc61580ff4072aa8f4eee437c91ce20298f09ae95d6cf9767e9ad89 74.24MB / 74.24MB

```

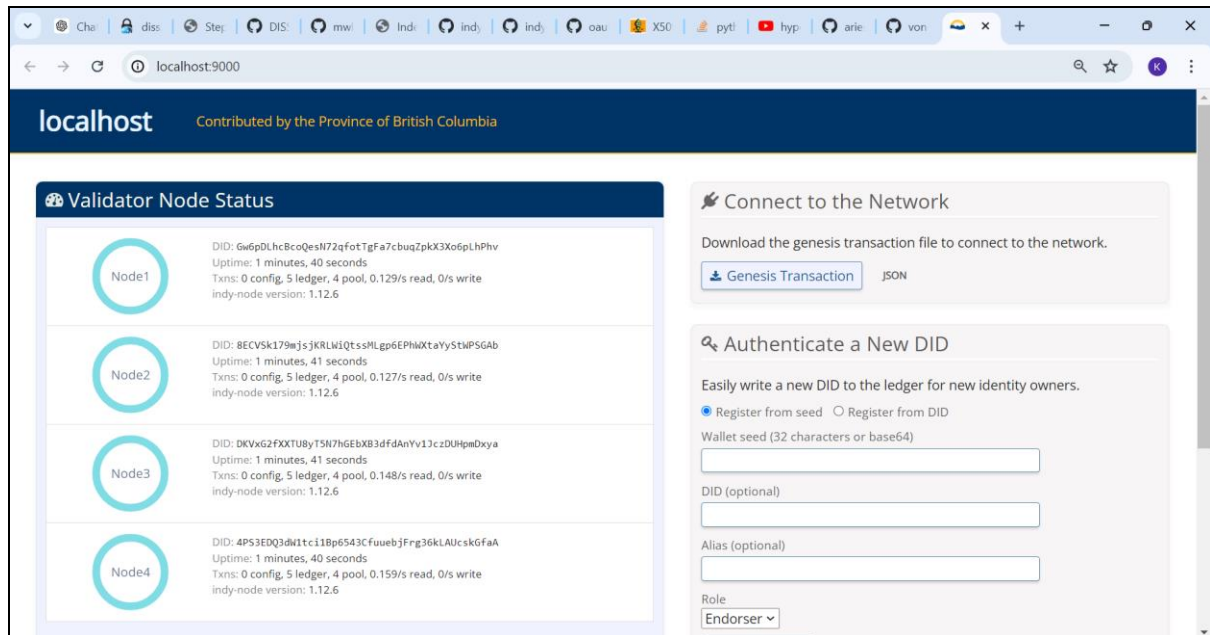
- Once build is successful, the network container can be seen on Docker Desktop like below:



- Once the ledger runs with this command

./manage build

you can check the ledger at – *http://localhost:9000*



- After public ledger is up and running, open a git bash terminal and git clone the Hyperledger Aries-Cloudagent-Python repo with below commands –

```
git clone https://github.com/hyperledger/aries-cloudagent-python
cd aries-cloudagent-python/demo
```

- Open another git bash terminal and change to this demo folder. Run the Faber agent with command –

```
./run_demo faber
```

```
MINGW64/C:/Users/ketkijaka x + v
ketki.jakatdar@Ketki MINGW64 ~
$ cd aries-cloudagent-python/demo

ketki.jakatdar@Ketki MINGW64 ~/aries-cloudagent-python/demo (main)
$ ./run_demo faber
Resolution not specified or invalid.
You can utilize the 'build' option to build the agent or the 'run' option to run the agent.
Preparing agent image...
[+] Building 6.4s (22/22) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 3.49kB
=> [internal] load metadata for docker.io/library/python:3.12-slim-bullseye
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 237B
=> [internal] load build context
=> => transferring context: 20.11MB
=> [main 1/12] FROM docker.io/library/python:3.12-slim-bullseye@sha256:11d30cce970c40aee8e993d302a15f8bc8204eca 0.0s
=> CACHED [main 2/12] RUN useradd -U -ms /bin/bash -u 1001 aries 0.0s
=> CACHED [main 3/12] RUN apt-get update -y && apt-get install -y --no-install-recommends apt-transport 0.0s
=> CACHED [main 4/12] WORKDIR /home/aries 0.0s
=> CACHED [main 5/12] RUN usermod -a -G 0 aries 0.0s
=> CACHED [main 6/12] RUN mkdir -p /home/aries/.aries_cloudagent /home/aries/.cache/pip/http /home/ 0.0s
=> CACHED [main 7/12] RUN chown -R aries:root /home/aries/.indy_client /home/aries/.aries_cloudagent && chm 0.0s
=> CACHED [main 8/12] RUN mkdir -p /home/indy 0.0s
=> CACHED [main 9/12] RUN ln -s /home/aries/.indy_client /home/indy/.indy_client 0.0s
=> CACHED [build 2/5] WORKDIR /src 0.0s
=> CACHED [build 3/5] ADD . . 0.0s
=> CACHED [build 4/5] RUN pip install --no-cache-dir poetry 0.0s
```

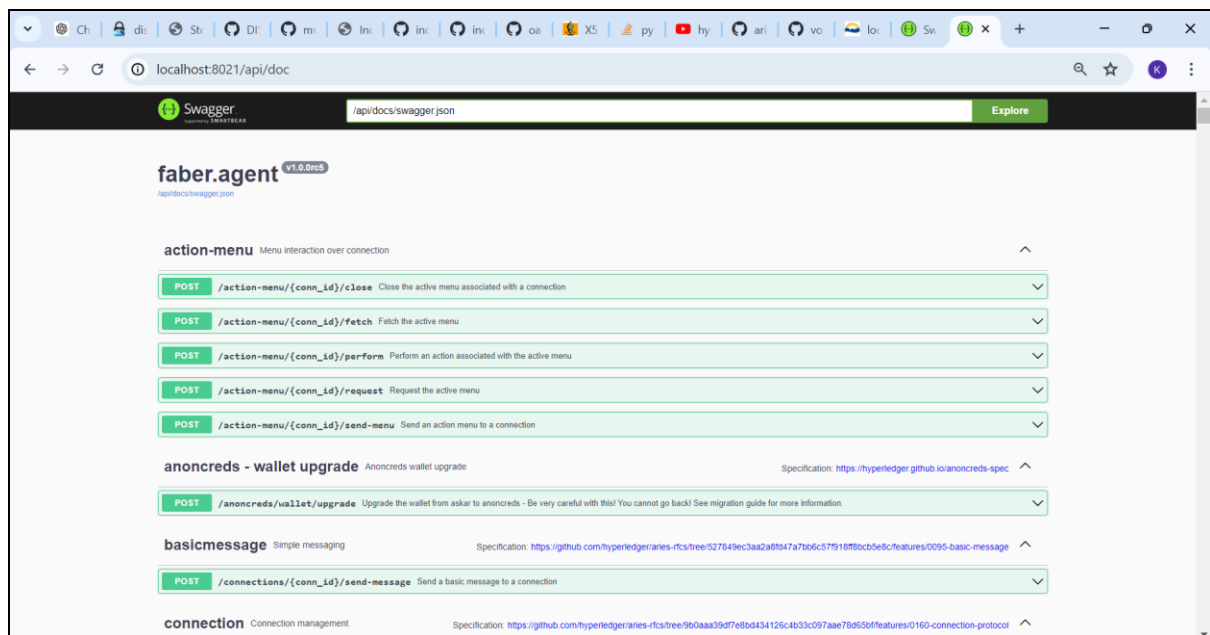
- Open a 3rd git bash terminal and change to this demo folder. Run the Alice agent with command –

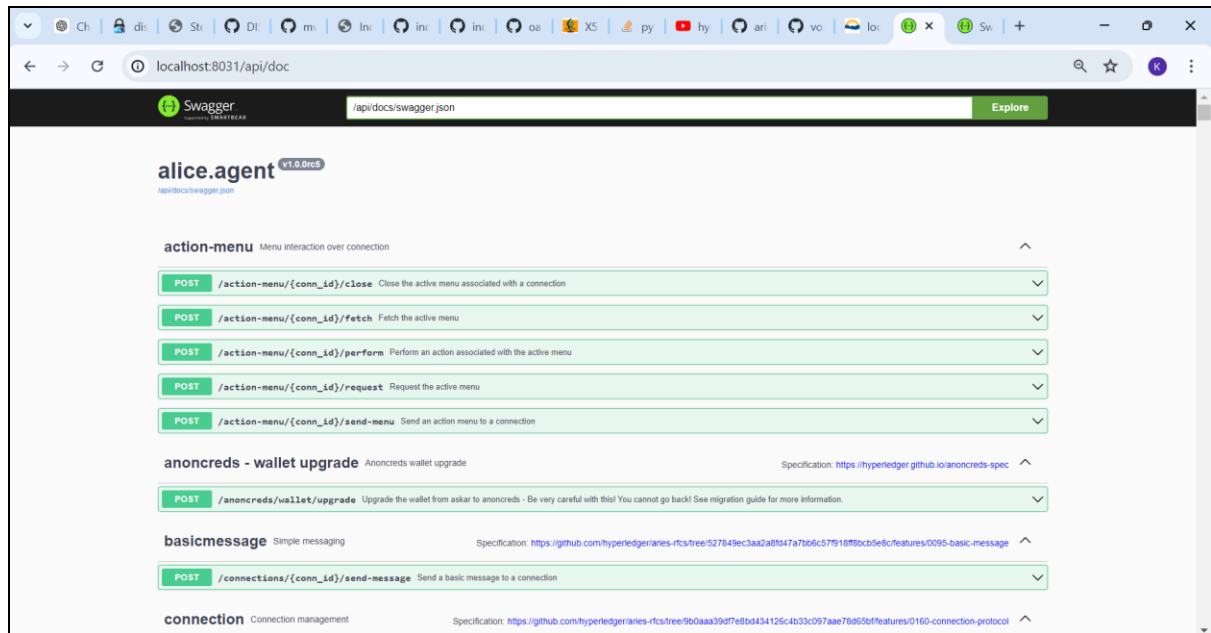
`./run_demo alice`

```
MINGW64/c/Users/ketkijaka x + v
ketki.jakatdar@Ketki MINGW64 ~
$ cd aries-cloudagent-python/demo

ketki.jakatdar@Ketki MINGW64 ~/aries-cloudagent-python/demo (main)
$ ./run_demo alice
Resolution not specified or invalid.
You can utilize the 'build' option to build the agent or the 'run' option to run the agent.
Preparing agent image...
[+] Building 1.4s (21/21) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 3.49kB                             0.0s
=> [internal] load metadata for docker.io/library/python:3.12-slim-bullseye 0.6s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 237B                                    0.0s
=> [build 1/5] FROM docker.io/library/python:3.12-slim-bullseye@sha256:11d30cce970c40aee8e993d302a15f8bc8204ecab 0.0s
=> [internal] load build context                                  0.6s
=> => transferring context: 1.56MB                                  0.5s
=> CACHED [main 2/12] RUN useradd -U -ms /bin/bash -u 1001 aries 0.0s
=> CACHED [main 3/12] RUN apt-get update -y && apt-get install -y --no-install-recommends apt-transport 0.0s
=> CACHED [main 4/12] WORKDIR /home/aries                          0.0s
=> CACHED [main 5/12] RUN usermod -a -G 0 aries                    0.0s
=> CACHED [main 6/12] RUN mkdir -p /home/aries/.aries_cloudagent /home/aries/.cache/pip/http /home/ 0.0s
=> CACHED [main 7/12] RUN chown -R aries:root /home/aries/.indy_client /home/aries/.aries_cloudagent && chm 0.0s
=> CACHED [main 8/12] RUN mkdir -p /home/indy                     0.0s
=> CACHED [main 9/12] RUN ln -s /home/aries/.indy_client /home/indy/.indy_client 0.0s
=> CACHED [build 2/5] WORKDIR /src                                0.0s
=> CACHED [build 3/5] ADD . .                                     0.0s
=> CACHED [build 4/5] RUN pip install --no-cache-dir poetry      0.0s
=> CACHED [build 5/5] RUN poetry build                            0.0s
```

- Once both agents are running, their admin consoles are accessible on browser at - <http://localhost:8021/> (Faber) and <http://localhost:8031/> (Alice) respectively.





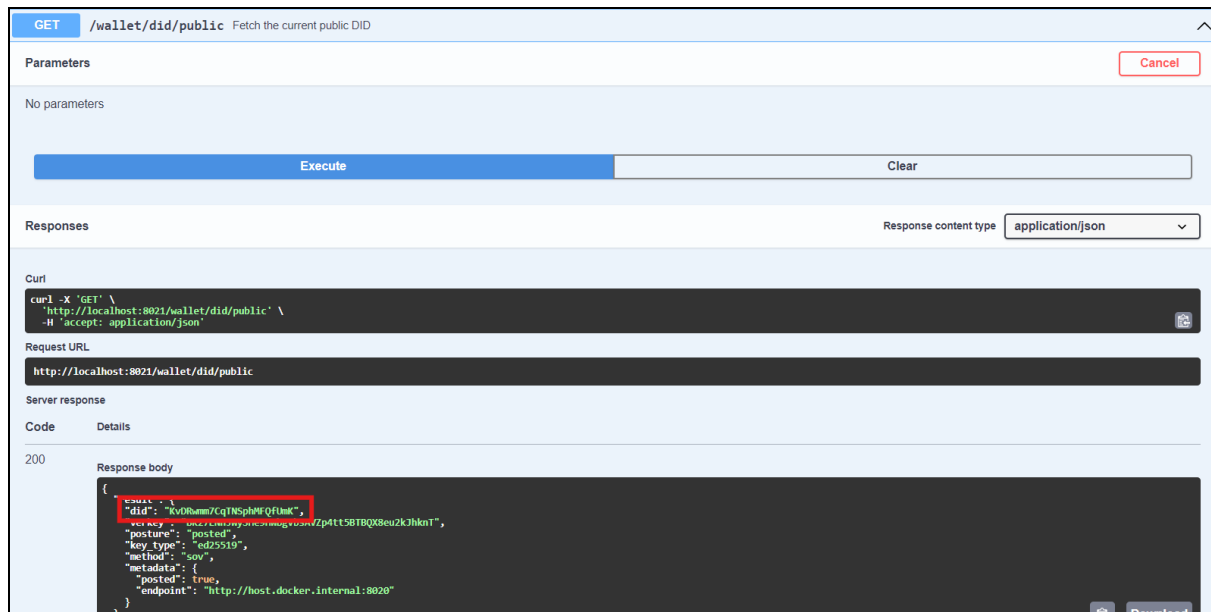
- Since we are using the demo scripts, Faber has already registered its public DID on the ledger, created a schema and a credential definition and registered them on the ledger.

```
Startup duration: 5.61s
Admin URL is at: http://host.docker.internal:8021
Endpoint URL is at: http://host.docker.internal:8020

#3/4 Create a new schema/cred def on the ledger
Schema:
{
  "sent": {
    "schema_id": "KvDRwmm7CqTNSphMFQfUmK:2:degree schema:77.33.75",
    "schema": {
      "ver": "1.0",
      "id": "KvDRwmm7CqTNSphMFQfUmK:2:degree schema:77.33.75",
      "name": "degree schema",
      "version": "77.33.75",
      "attrNames": [
        "date",
        "degree",
        "birthdate_dateint",
        "timestamp",
        "name"
      ],
      "seqNo": 8
    }
  },
  "schema_id": "KvDRwmm7CqTNSphMFQfUmK:2:degree schema:77.33.75",
  "schema": {
    "ver": "1.0",
    "id": "KvDRwmm7CqTNSphMFQfUmK:2:degree schema:77.33.75",
    "name": "degree schema",
    "version": "77.33.75",
    "attrNames": [
      "date",
      "degree",
      "birthdate_dateint",
      "timestamp",
      "name"
    ],
    "seqNo": 8
  }
}
```

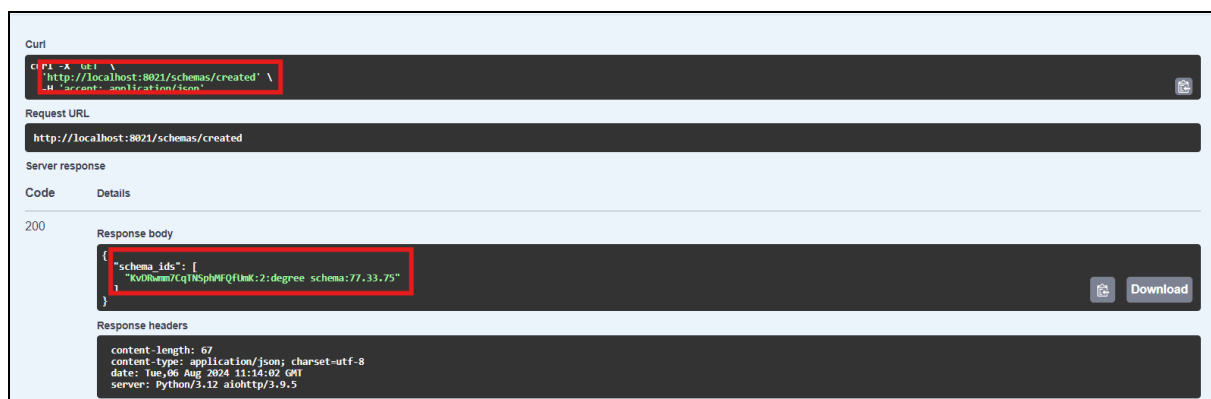
- First get Faber's public DID from endpoint –

GET /wallet/did/public



- The schema can be retrieved from below endpoints –

GET /schemas/created



- Note this schema id and paste it in endpoint below endpoint to get the schema:

GET /schemas/{schema_id}

GET /schemas/{schema_id} Gets a schema from the ledger

Parameters

Name	Description
schema_id * required	Schema identifier

string (path)

KvDRwmm7CqTNSphMFQfUmK:2:degree s

Execute Clear

Responses

Response content type application/json

Curl

```
curl -X 'GET' \
  'http://localhost:8021/schemas/KvDRwmm7CqTNSphMFQfUmK:2:degree schema:77.33.75' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8021/schemas/KvDRwmm7CqTNSphMFQfUmK:2:degree schema:77.33.75

Server response

Code Details

200

Response body

```
{
  "schema": {
    "ver": "1.0",
    "id": "KvDRwmm7CqTNSphMFQfUmK:2:degree schema:77.33.75",
    "name": "degree schema",
    "attrNames": [
      "date",
      "degree",
      "name",
      "birthdate_dateint",
      "timestamp"
    ]
  }
}
```

Download

- Similarly get credentials definition:

GET /credential-definitions/created

Curl

```
curl -X 'GET' \
  'http://localhost:8021/credential-definitions/created' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8021/credential-definitions/created

Server response

Code Details

200

Response body

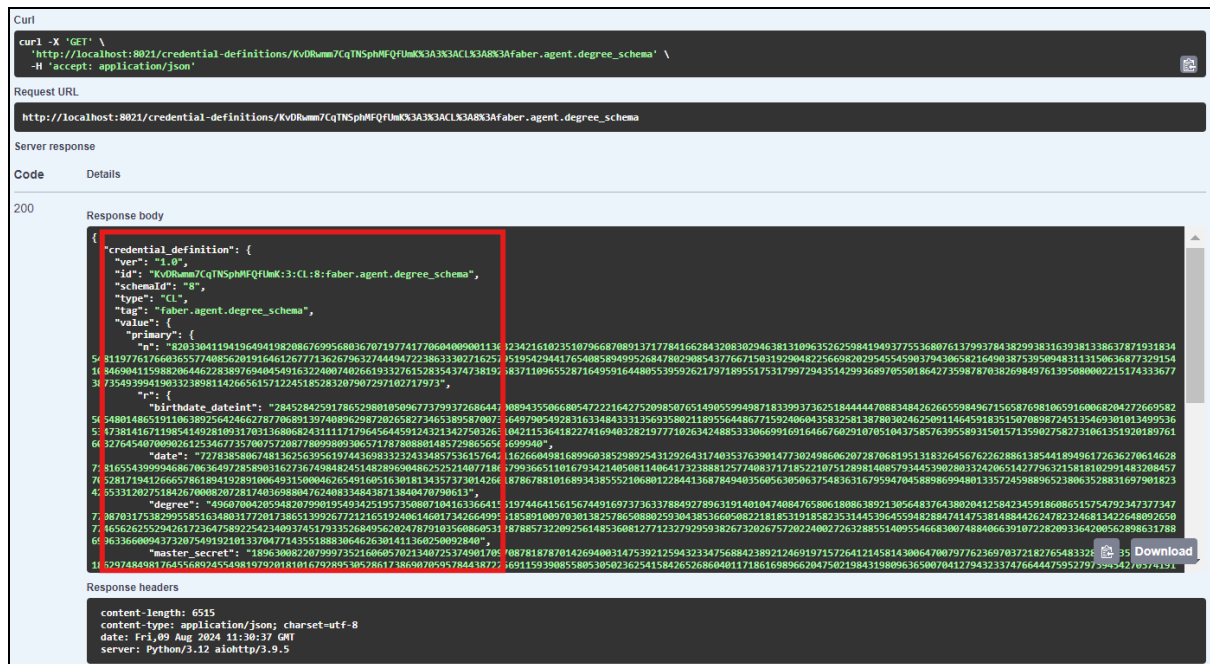
```
{
  "credential_definition_ids": [
    "KvDRwmm7CqTNSphMFQfUmK:3:cl:8:faber.agent.degree_schema"
  ]
}
```

Response headers

```
content-length: 99
content-type: application/json; charset=utf-8
date: Fri, 09 Aug 2024 11:29:12 GMT
server: Python/3.12 aiohttp/3.9.5
```

Download

GET /credential-definitions/{cred_def_id}



3.2 ACE-OAuth server

- Download and install Composer v2.7.7 from [HERE](https://getcomposer.org/download/)⁵.
- The standard OAuth server is available [HERE](https://github.com/bshaffer/oauth2-server-php)⁶. To modify this standard server into ACE-OAuth server, we first need to setup Apache because we need to enable Server Name Indication (SNI) and it cannot be done on built-in PHP server which comes with the standard OAuth server. Unzip the given artifact, extract the “*httpd-2.4.62-240718-win64-VS17*” directory and place it at location “*C:/*” location.
- All the necessary changes in the files *httpd.conf*, *httpd-vhosts.conf*, *httpd-ssl.conf* along with SSL certificates and SNI configuration, is present in this directory.
- Download and Install Visual Studio Code IDE 1.92.0 x64 from [HERE](https://code.visualstudio.com/docs/setup/windows)⁷.
- Clone the standard OAuth server using below commands:

```
mkdir oauth2-server-1
cd oauth2-server-1
git clone https://github.com/bshaffer/oauth2-server-php.git -b main
```

- From the extracted artifact, copy and paste the files from “*oauth2-server-1*” folder to below locations in the newly created directory.

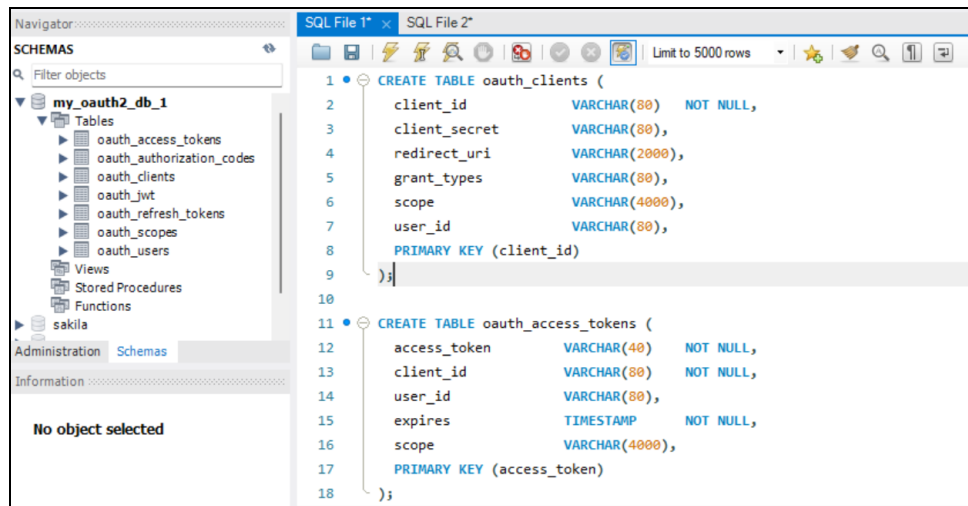
⁵ <https://getcomposer.org/download/>

⁶ <https://github.com/bshaffer/oauth2-server-php>

⁷ <https://code.visualstudio.com/docs/setup/windows>

File name	Relative path location
<i>server.php</i>	..\oauth2-server-1\oauth2-server-php\
<i>token.php</i>	..\oauth2-server-1\oauth2-server-php\
<i>Config.php</i>	..\oauth2-server-1\oauth2-server-php\src\OAuth2\
<i>DIDGrant.php</i>	..\oauth2-server-1\oauth2-server-php\src\OAuth2\GrantType\

- Download and install MySQL Workbench 8.0.38 from [HERE](#)⁸.
- Open MySQL Workbench app and create a new schema “*my_oauth2_db_1*”. Select the schema and paste the commands available from [HERE](#)⁹ under the “*Define your Schema*” section and hit Run.

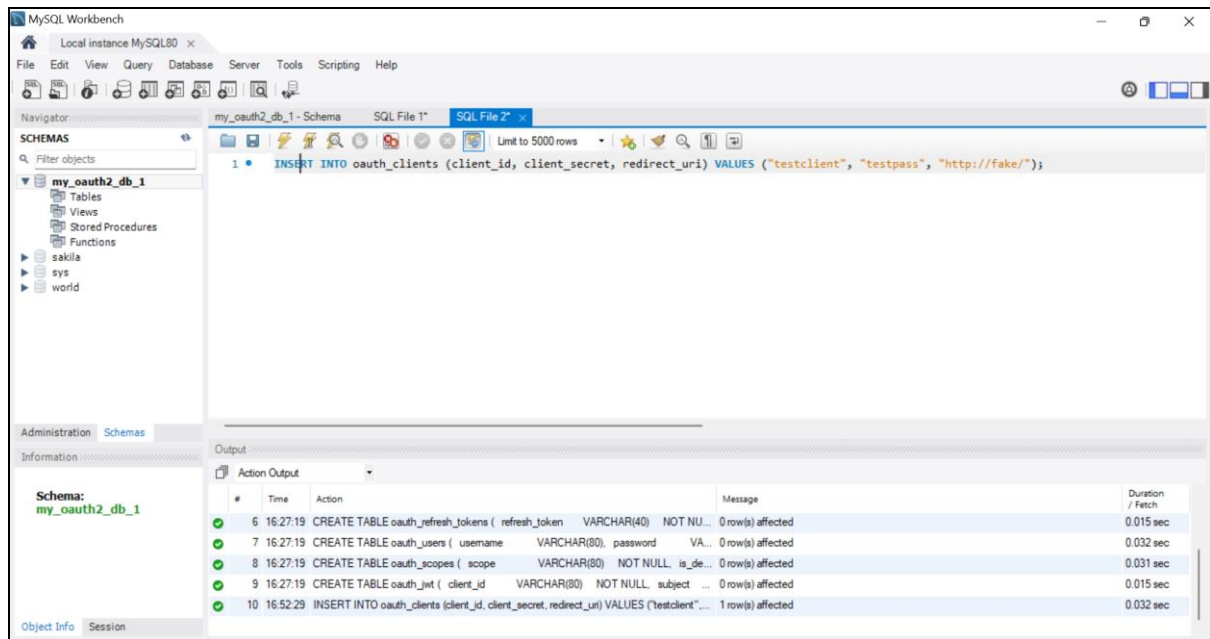


- Run the following command to create an OAuth test client:

```
INSERT INTO oauth_clients (client_id, client_secret, redirect_uri)
VALUES ("testclient", "testpass", "http://fake/");
```

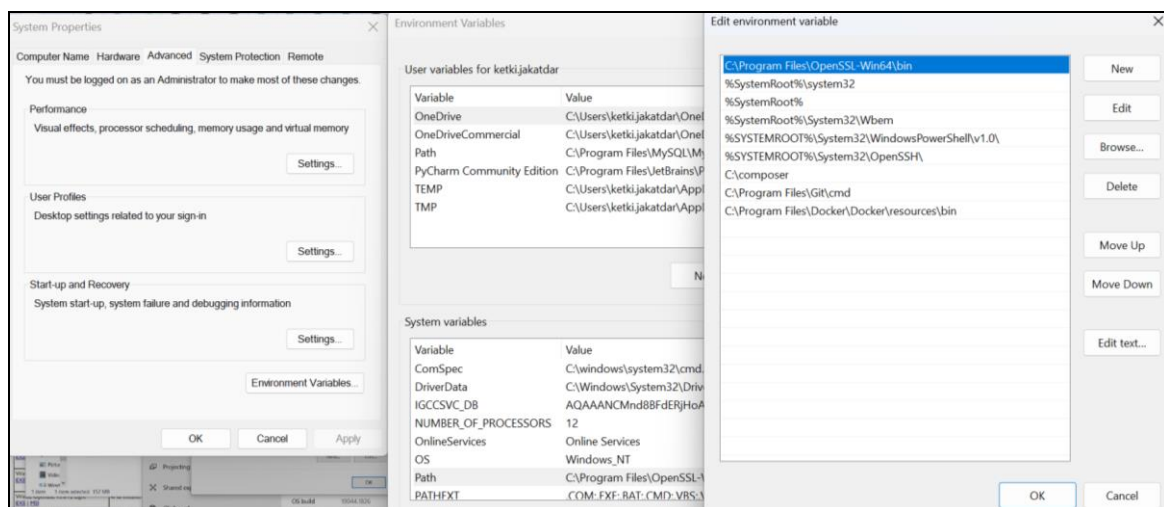
⁸ <https://dev.mysql.com/downloads/windows/installer/>

⁹ <https://bshaffer.github.io/oauth2-server-php-docs/cookbook/>



3.3 OpenSSL

- Download and install OpenSSL v3.3.1 from [HERE](#)¹⁰.
- After finishing the installation, add the path of bin folder in PATH environment variable and System variable.



3.4 Resource Server (Printer)

- Download Node.js and NPM installer from [HERE](#)¹¹.
- Initialize a new Node.js project with below command. This will create a “package.json” file with default values. Next install express.

¹⁰ <https://slproweb.com/products/Win32OpenSSL.html>

¹¹ <https://nodejs.org/en/download/prebuilt-installer>

```
npm init -y
npm install express typescript ts-node-dev @types/node
@types/express
```

```
PS C:\Users\ketki.jakatdar> cd C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server
PS C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server> npm init -y
Wrote to C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server\package.json:

{
  "name": "resource-server",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}

PS C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server> npm install express typescript ts-node-dev @types/node @types/express
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated rimraf@2.7.1: Rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 137 packages, and audited 138 packages in 21s

20 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- Install necessary packages for json handling

```
npm install express ejs body-parser jsonwebtoken
npm install --save-dev @types/jsonwebtoken
```

```
PS C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server> npm install express ejs body-parser jsonwebtoken
added 26 packages, and audited 164 packages in 2s

22 packages are looking for funding
  run `npm fund` for details

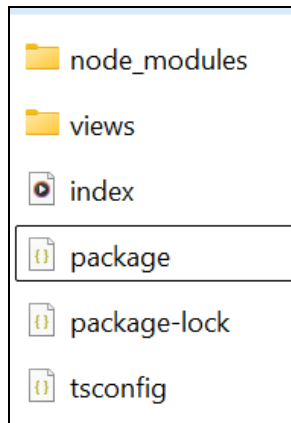
found 0 vulnerabilities

PS C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server> npm install --save-dev @types/jsonwebtoken
added 1 package, and audited 165 packages in 2s

22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- Move the “*index.ts*” and “*tsconfig.json*” files from the “*Resource Server*” directory from the artifact provided, into your current directory. The folder structure should look something like this:



- Install typescript globally

npm install -g typescript

```
PS C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server> npm install -g typescript
added 1 package in 2s
```

3.5 Postman (client), Wireshark, Burp Suite

- Download and install Postman v11.7.0 from [HERE](#)¹². Create & login with a test account.
- Download and install Wireshark 4.2.6 version from [HERE](#)¹³. Launch the app.
- Download and install Burp Suite v2024.5.5 from [HERE](#)¹⁴. Open the app with temporary project and click Start.

4 Implementation

4.1 Issue DID & VC to Alice from Faber

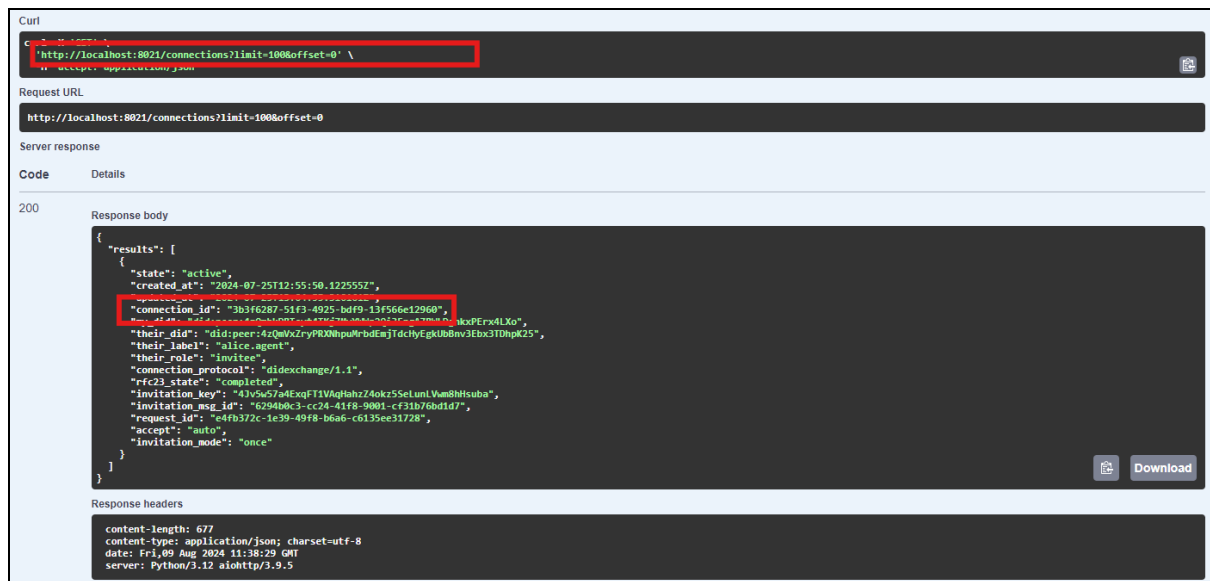
- Get Faber's connection id –

GET /connections

¹² <https://www.postman.com/downloads/>

¹³ <https://www.wireshark.org/download.html>

¹⁴ <https://portswigger.net/burp/releases/professional-community-2024-5-5>



- Issue Credential (DID & VC). From Faber agent modify the JSON file like below: (Get the attributes from HERE¹⁵)

POST /issue-credential-2.0/send

POST /issue-credential-2.0/send Send holder a credential, automating entire flow

Parameters

Name	Description
body object (body)	Edit Value Model

```
{
  "auto_remove": true,
  "connection_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "credential_preview": {
    "@type": "issue-credential/2.0/credential-preview",
    "attributes": [
      {
        "name": "name",
        "value": "Alice Smith"
      },
      {
        "name": "timestamp",
        "value": "1234567890"
      },
      {
        "name": "date",
        "value": "2018-05-28"
      },
      {
        "name": "degree",
        "value": "Maths"
      },
      {
        "name": "birthdate_dateint",
        "value": "19640101"
      }
    ]
  },
  "filter": {
    "in": [
      {
        "cred_def_id": "KvDRumm7CqTNSphMFQfUmK:3:CL:8:faber.agent.degree_schema",
        "schema_id": "KvDRumm7CqTNSphMFQfUmK",
        "schema issuer did": "KvDRumm7CqTNSphMFQfUmK:2:degree_schema:77.33.75",
        "schema name": "degree schema",
        "schema version": "77.33.75"
      }
    ]
  },
  "id proof": {
    "credential": {
      "@context": [
        "https://www.w3.org/2018/credentials/v1",
        "https://w3id.org/citizenship/v1"
      ],
      "credentialSubject": {
```

¹⁵ <https://aca-py.org/latest/demo/AriesOpenAPIDemo/#faber-issuing-the-credential>

Responses

Curl

```
curl -X 'POST' \
'http://localhost:8021/issue-credential-2.0/send' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "auto_remove": true,
  "comment": "string",
  "connection_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "credential_preview": {
    "@type": "issue-credential/2.0/credential-preview",
    "attributes": [
      {
        "name": "name",
        "value": "Alice Smith"
      },
      {
        "name": "timestamp",
        "value": "1234567890"
      },
      {
        "name": "date",
        "value": "2018-05-28"
      },
      {
        "name": "degree",
        "value": "Maths"
      },
      {
        "name": "birthdate_dateint",
        "value": "19640101"
      }
    ]
  },
  "filter": {
    "indy": {
      "cred_def_id": "KvDRwmn7CqTNSphMFQfUmK:3:CL:8:faber.agent.degree_schema",
      "issuer_did": "KvDRwmn7CqTNSphMFQfUmK",
      "schema_id": "KvDRwmn7CqTNSphMFQfUmK:2:degree schema:77.33.75",
      "schema_issuer_did": "KvDRwmn7CqTNSphMFQfUmK",
      "schema_name": "degree schema",
      "schema_version": "77.33.75"
    },
    "id_proof": {
      "credential": {
        "@context": [
          "https://www.w3.org/2018/credentials/v1",
          "https://w3id.org/citizenship/v1"
        ]
      }
    }
  }
}
```

- Move to Alice's agent in gitbash. Alice agent's source code automatically handles the credential issuance request. It first receives the credential offer, responds to it with a request to Faber, Faber receives the request and finally offer the credentials.

```
Connect duration: 0.14s
Waiting for connection...
Alice      | Connected
Alice      | Check for endorser role ...
Connect duration: 0.53s
Alice      | Credential: state = offer-received, cred_ex_id = 08b2d69b-194d-4562-9645-4d7742e51247

#15 After receiving credential offer, send credential request
Alice      | Credential: state = request-sent, cred_ex_id = 08b2d69b-194d-4562-9645-4d7742e51247
Alice      | Credential: state = credential-received, cred_ex_id = 08b2d69b-194d-4562-9645-4d7742e51247
```

- Now, Alice needs to save the credentials in her wallet. On Alice Swagger UI get '*cred_ex_id*' from the endpoint –

GET /issue-credential-2.0/records

Curl

```
curl -X 'GET' \
'http://localhost:8031/issue-credential-2.0/records?limit=100&offset=0' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8031/issue-credential-2.0/records?limit=100&offset=0
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "results": [{ "cred_ex_record": { "state": "done", "created_at": "2024-07-25T13:10:46.244173Z", "updated_at": "2024-07-25T13:10:46.931306Z", "state": "raise", "cred_ex_id": "08b2d69b-194d-4562-9645-4d7742e51247", "connection_id": "0c0f47ea-f4c3-43d6-8851-2c01263b80a1", "thread_id": "0edbcd3d-66c2-495f-8a2d-b776bfdbbecf", "initiator": "external", "role": "holder", "cred_offer": { "@type": "https://didcomm.org/issue-credential/2.0/offer-credential", "@id": "0edbcd3d-66c2-495f-8a2d-b776bfdbbecf", "~thread": {}, "comment": "Offer on cred def id KvDRwmm7CqTNSphMFQfUmK:3:CL:8:faber.agent.degree_schema", "credential_preview": { "@type": "https://didcomm.org/issue-credential/2.0/credential-preview", "attributes": [{ "name": "name", "value": "Alice Smith" }, { "name": "date", "value": "2018-05-28" }] } } } }] }</pre>

POST /issue-credential-2.0/records/{cred_ex_id}/store

As Alice's credentials were already store in wallet received error. Otherwise, no error.

cred_ex_id * required
string (path)

Credential exchange identifier

08b2d69b-194d-4562-9645-4d7742e51247

Execute Clear

Responses Response content type application/json

Curl

```
curl -X 'POST' \
'http://localhost:8031/issue-credential-2.0/records/08b2d69b-194d-4562-9645-4d7742e51247/store' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "credential_id": "string"
}'
```

Request URL

```
http://localhost:8031/issue-credential-2.0/records/08b2d69b-194d-4562-9645-4d7742e51247/store
```

Server response

Code	Details
400	<p>Undocumented Error: Credential exchange 08b2d69b-194d-4562-9645-4d7742e51247 in done state (must be credential-received).</p> <p>Response body</p> <pre>400: Credential exchange 08b2d69b-194d-4562-9645-4d7742e51247 in done state (must be credential-received).</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: http://localhost:8031 access-control-expose-headers: Content-Length,Server,Date content-length: 186 content-type: text/plain; charset=utf-8 date: Fri, 09 Aug 2024 12:08:59 GMT server: Python/3.12 aiohttp/3.9.5</pre>

To confirm the credentials were stored –

GET /credentials



- Go to Faber agent's terminal. It received confirmation that the credentials were received and accepted.

```
#13 Issue credential offer to X
Faber | Credential: state = offer-sent, cred_ex_id = 8c16ea4a-07aa-423d-b3e9-447bdb1d835b
Faber | Credential: state = request-received, cred_ex_id = 8c16ea4a-07aa-423d-b3e9-447bdb1d835b

#17 Issue credential to X
Faber | Credential: state = credential-issued, cred_ex_id = 8c16ea4a-07aa-423d-b3e9-447bdb1d835b
Faber | Credential: state = done, cred_ex_id = 8c16ea4a-07aa-423d-b3e9-447bdb1d835b
```

4.2 Generate proof embedded X509 certificates

All the files created in this step are provided in the artifact under “*Certificates*” folder. Below steps were executed to create all these files.

4.2.1 X509 certificate with custom extensions for Alice

1. Set Up Certificate Authority (CA)

- Generate CA private key

[illegible]

- Create CA cert


```
ketki.jakatdar@Ketki MINGW64 ~/Desktop/Thesis/Code/Certificate/With_ext
$ openssl req -new -key alice.key -out alice.csr -config csr.cnf
```

3. Create a Custom Configuration File for Extensions

- Create “*ext.cnf*” like below:

```
[ req ]
default_bits = 2048
prompt = no
default_md = sha256
distinguished_name = dn
req_extensions = v3_req

[ dn ]
C = IR
ST = Dublin
L = Dublin
O = NCI
OU = Cybersecurity
CN = localhost

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = digitalSignature, keyEncipherment
subjectAltName = @alt_names

# Custom extensions with key-value pairs
1.3.6.1.4.1.12345.1 = ASN1:UTF8String:DID: 4zQmVxZryPRXNhpuMrbdEmjTdcHyEgkUbBnv3Ebx3TDhpK25
1.3.6.1.4.1.12345.2 = ASN1:UTF8String:Name: Alice Smith
1.3.6.1.4.1.12345.3 = ASN1:UTF8String:Degree: Maths
1.3.6.1.4.1.12345.4 = ASN1:UTF8String:Date: 2018-05-28
1.3.6.1.4.1.12345.5 = ASN1:UTF8String:Birthdate: 20000725
1.3.6.1.4.1.12345.6 = ASN1:UTF8String:Timestamp: 1721913045
1.3.6.1.4.1.12345.7 = ASN1:UTF8String:ResourceServer: Printing Server 1

[ alt_names ]
DNS.1 = localhost
```

- Sign Alice’s cert with CA & “*csr.cnf*” and include extensions from “*ext.cnf*”’s “*v3_req*” section

```
ketki.jakatdar@Ketki MINGW64 ~/Desktop/Thesis/Code/Certificate/With_ext
$ openssl x509 -req -in alice.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out alice.crt -days 365 -extfile ext.cnf -extensions v3_req
Certificate request self-signature ok
subject=C=IR, ST=Dublin, L=Dublin, O=NCI, OU=Cybersecurity, CN=localhost
```

- Create “*alice.pfx*” file by combining “*alice.key*” and “*alice.crt*”. Postman accepts .pfx file.

```
ketki.jakatdar@Ketki MINGW64 ~/Desktop/Thesis/Code/Certificate/With_ext
$ openssl pkcs12 -export -out alice.pfx -inkey alice.key -in alice.crt -certfile ca.crt
Enter Export Password:

Verifying - Enter Export Password:
```

- Below are all the files created for Alice:

ca.pem	PEM File
alice	Personal Information Exchange
alice	Security Certificate
ca.srl	SRL File
alice.csr	CSR File
csr.cnf	CNF FILE
ext.cnf	CNF FILE
alice.key	KEY File
ca	Security Certificate
ca.key	KEY File

4.2.2 SSL certificates for Authorization Server

4. Create self-signed certificate and key for Authorization Server (AS)

These certificates don't have custom extensions.

Create private key for CA. Create Root Certificate for the CA.

```

Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ketki.jakatdar>openssl version
OpenSSL 3.3.1 4 Jun 2024 (Library: OpenSSL 3.3.1 4 Jun 2024)

C:\Users\ketki.jakatdar>set CANAME=MyOrg-RootCA

C:\Users\ketki.jakatdar>openssl genrsa -aes256 -out %CANAME%.key 4096
Enter PEM pass phrase:

Verifying - Enter PEM pass phrase:

C:\Users\ketki.jakatdar>openssl req -x509 -new -nodes -key %CANAME%.key -sha256 -days 1826 -out %CANAME%.crt
Enter pass phrase for MyOrg-RootCA.key:

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IR
State or Province Name (full name) [Some-State]:Dublin
Locality Name (eg, city) []:Dublin
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NCI
Organizational Unit Name (eg, section) []:Cybersecurity
Common Name (e.g. server FQDN or YOUR name) []:CS
Email Address []:jakatdarketki@gmail.com

C:\Users\ketki.jakatdar>

```

- Create “*MyServer.key*” and “*MyServer.crt*” self-signed certificates for the AS signed by above Root certificate.

```

Command Prompt
C:\Users\ketki.jakatdar>set MYCERT=MyServer









C:\Users\ketki.jakatdar>openssl req -new -nodes -out %MYCERT%.csr -newkey rsa:4096 -keyout %MYCERT%.key
.....
Certificate request self-signature ok
subject=C=IR, ST=Dublin, L=Dublin, O=NCI, OU=Cybersecurity, CN=CS, emailAddress=jakatdarketki@gmail.com
Enter pass phrase for MyOrg-RootCA.key:
-----

C:\Users\ketki.jakatdar>notepad %MYCERT%.v3.ext

C:\Users\ketki.jakatdar>openssl x509 -req -in %MYCERT%.csr -CA %CANAME%.crt -CAkey %CANAME%.key -CAcreateserial -out %MYCERT%.crt -days 730 -sha256 -extfile %MYCERT%.v3.ext
Certificate request self-signature ok
subject=C=IR, ST=Dublin, L=Dublin, O=NCI, OU=Cybersecurity, CN=CS, emailAddress=jakatdarketki@gmail.com
Enter pass phrase for MyOrg-RootCA.key:

```

- Below is the list of certificates created for the AS

 MyOrg-RootCA.key	KEY File
 MyOrg-RootCA	Security Certificate
 MyServer.key	KEY File
 MyServer.csr	CSR File
 MyServer.v3.ext	EXT File
 MyOrg-RootCA.srl	SRL File
 MyServer	Security Certificate
 MyServer.pem	PEM File

4.3 Launch the ACE-OAuth server

- To start Apache server, open Command Prompt as an administrator. Navigate to “C:\httpd-2.4.62-240718-win64-VS17\Apache24\bin” & run below command –

To start the server → *httpd.exe -k start*

To stop the server → *httpd.exe -k stop*

To restart the server → *httpd.exe -k restart*

```
C:\httpd-2.4.62-240718-win64-VS17\Apache24\bin>httpd.exe -k start  
C:\httpd-2.4.62-240718-win64-VS17\Apache24\bin>_
```

- You can access the server on browser at – <https://localhost:443>. You'll get a warning about the certificate because it is a self-signed certificate.

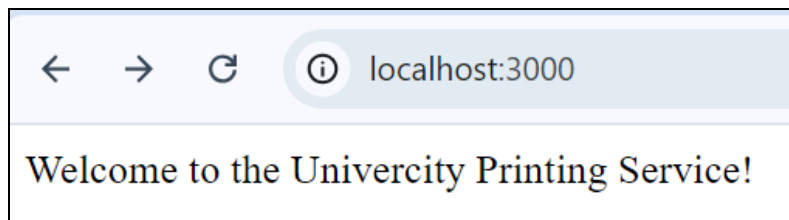


4.4 Launch the Resource Server (Printer)

- Open another gitbash terminal or Command Prompt and change to the directory of the Resource Server. Compile “*index.ts*” into “*index.js*” and launch the Resource Server and access it at – <http://localhost:3000>.

```
tsc  
node dist/index.js
```

```
PS C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server> tsc  
PS C:\Users\ketki.jakatdar\Desktop\Thesis\Code\Resource-Server> node dist/index.js  
Resource Server running on port 3000
```



4.5 Configure Postman requests

- Launch Postman app → Navigate to Settings → Go to the Certificates Tab → Select the Certificates tab on the left sidebar → Add a Client Certificate → Click Add Certificate → Configure the Certificate like below:

Host: localhost.

CRT file: Choose the “*alice.pfx*, *alice.crt*, *alice.key*” files you created.

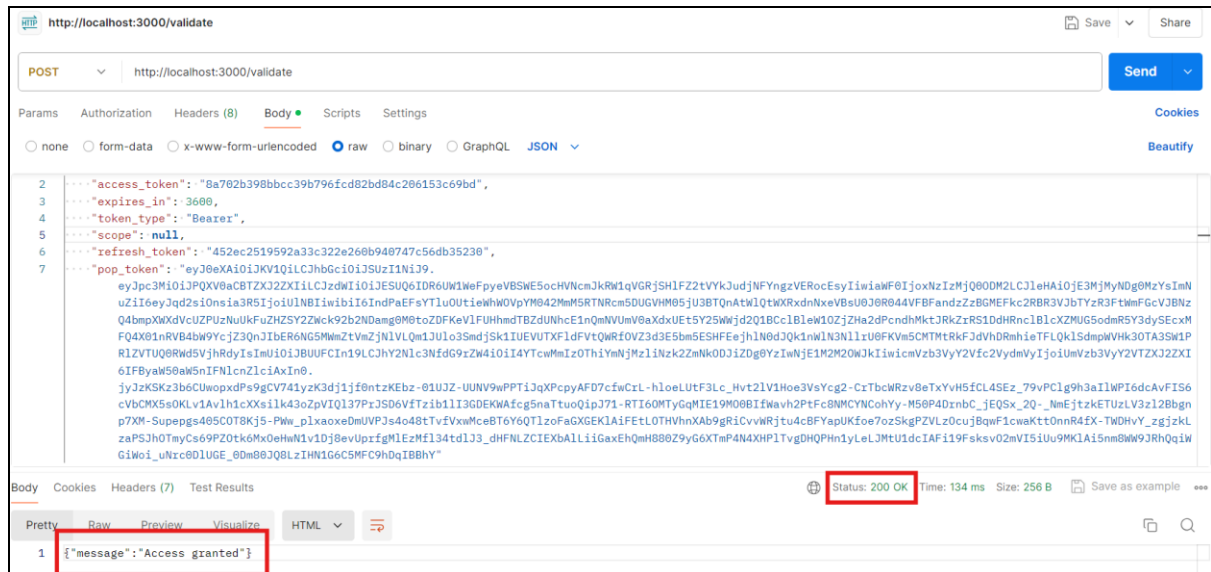
Passphrase: Enter the passphrase if you set one during the export (if not, leave it blank)

Now, when you send the request, POSTMAN will automatically use the client certificate associated with the host and port combination you've specified.

- Create a new POST request to send to the AS. Endpoint is – ***“https://localhost:token.php”***. Modify the body of the request as below:

- Hit Send. Below is the response which contains the Proof of Possession (PoP) token.

- Create another POST HTTP request to send to the RS. Endpoint is – “<http://localhost:3000/validate>”. Copy the entire response json from AS and paste it in the body of this request. Hit Send.



5 Security Testing

5.1 PoP token signature verification

- Open jwt.io¹⁶ in browser. Select “**RSA256**” in the Algorithm dropdown.
- Copy only the encoded “**pop_token**” from the response received from the AS and paste it in the “**Encoded**” column on jwt.io. It will automatically decode the token and the decoded body and header of the token are visible in the “**Decoded**” column on the right side.



¹⁶ <https://jwt.io/>

Algorithm
RS256

Encoded

PASTE A TOKEN HERE

```

UGVHM05jU3BTQnAtWlQtWXRxdnNxeVBsU0J0R04
4VFBFandzZzBGMEFkc2RBR3VJbTYzR3FtWmFGcV
JBNzQ4bnpXWXdVcUzPUzNuUkFuZHZSY2ZWck92b
2NDamg0M0toZDFKeV1FUHmdTBZdUNhcE1nQmNV
UmV0aXdxUet5Y25WWjd2Q1BCc1BleW10ZjZHa2d
PcndhMktJrKzRS1DdHRnc1B1cXZMUG5odmR5Y3
dySEcxMFQ4X01nRVB4bW9YcjZ3QnJiER6NG5MW
mZtVmZjN1VLQm1JU1o3SmdjSk1IEUVtXF1dFVt
QWRfOVZ3d3E5bm5ESHFEejh1N0dJQk1nW1N3N1l
rU0KVM5CMTmtRkFjdVhDRmhieTFLQk1SdmpVWH
k30TA3SW1PR1ZVTUQ0RwD5VjhRdyIsImUiOiJBU
UFCIn19LCJhY2Nlc3NfdG9rZW4iOiI4YTcwMmIz
OThiYmNmZ21iNzk2ZmNkODJiZDg0YzIwNjE1M2M
2OWJkIiwicmVzbnV5Y2Vfc2VydmlvYjoiUmVzbnV5
Y2V2VTZXJ2Z2X1E6IFByaW50aW5nIFN1cnZlc1AxI
n0.jyZjKSKz3b6CUwopxdPs9gCV741yzK3dj1jf
0ntzKEbz-01UJZ-
UUNV9wPPTiJqXpPyAFD7cfwCrL-
hloeLUtF3Lc_Hvt2L1V1Hoe3VsYcg2-
CrTbcWRzv8eTxYvH5fCL4SEz_79vPC1g9h3aI1W
PI6dcAvFIS6cVbCMX5sOKLv1Av1h1cXXsilk43o
ZpVIQ137PrJSD6VftZib1I3GDEKWAfcg5naTtu
oQipJ71-
RTI60MTyGqMIE19M00BIfWavh2PtFc8NMCYNCo
Yy-M50P4DrnbC_jEQSx_2Q-

```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```

{
  "typ": "JWT",
  "alg": "RS256"
}

```

PAYLOAD: DATA

```

{
  "iss": "OAuth Server",
  "sub": "OID:",
  "4zQmVxZryPRXNhpMrbdEmjTdcHyEgkUbBnv3Ebx3TDhpK2",
  "iat": 1723244836,
  "exp": 1723244836,
  "cnf": {
    "jwk": {
      "kty": "RSA",
      "n": "wOhA1a9n9KbyhV9ZXN62c9E3QrnCPeG3NcSpS8p-
ZT-
YtqvsgyP1SBtGN8TPEjwsg0BfAdsdAGuIm63GmZeFqRA748njWYwUq
FOS3nRAndvRcFvRovocCjh43Khd1JyYpXfu8YuCapMgBcURetiwwPK
ycnVZ7vCPBRPeymNf6Gkg0rwa2K1FFKE-
CttgrPeqvLPhndycwrHG10T8_MgEPxmoXr6wBrH1Dz4nLZfmVfc6UK
BmIRZ7JgcJMPETMqetUmAd_9Vwwq9nnDhgDz8e7GIBMqZSw6YkSAJV
nB13-FA1uXCFby1K8IRvjVtY7987ImOFVUMD4EgyV8Qw",
      "e": "AQAB"
    }
  },
  "access_token":
    "8a782b398bbcc39b796fcd82bd84c206153c69bd",
  "resource_server": "ResourceServer: Printing Server"
}

```

- However, we see **“Invalid Signature”** at the bottom.

Algorithm
RS256

Encoded

PASTE A TOKEN HERE

```

TNRCm5DUGVHM05jU3BTQnAtWlQtWXRxdnNxeVB
sU0J0R044VFBFandzZzBGMEFkc2RBR3VJbTYzR
3FtWmFGcVJBNzQ4bnpXWXdVcUzPUzNuUkFuZHZ
SY2ZWck92b2NDamg0M0toZDFKeV1FUHmdTBZd
UNhcE1nQmNVUmV0aXdxUet5Y25WWjd2Q1BCc1B
leW10ZjZHa2dPcndhMktJrKzRS1DdHRnc1B1c
XZMUG5odmR5Y3dySEcxMFQ4X01nRVB4bW9YcjZ
3QnJiER6NG5MWmZtVmZjN1VLQm1JU1o3SmdjS
k1IEUVtXF1dFVtQWRfOVZ3d3E5bm5ESHFEejh
1N0dJQk1nW1N3N1lru0KVM5CMTmtRkFjdVhDR
mhieTFLQk1SdmpVWHk30TA3SW1PR1ZVTUQ0RwD
5VjhRdyIsImUiOiJBUUFCIn19LCJhY2Nlc3Nfd
G9rZW4iOiI4YTcwMmIzOThiYmNmZ21iNzk2ZmN
kODJiZDg0YzIwNjE1M2M2OWJkIiwicmVzbnV5Y
2Vfc2VydmlvYjoiUmVzbnV5Y2V2VTZXJ2Z2X1E6
IFByaW50aW5nIFN1cnZlc1AxIn0.jyZjKSKz3b6CU
wopxdPs9gCV741yzK3dj1jf0ntzKEbz-01UJZ-
UUNV9wPPTiJqXpPyAFD7cfwCrL-
hloeLUtF3Lc_Hvt2L1V1Hoe3VsYcg2-
CrTbcWRzv8eTxYvH5fCL4SEz_79vPC1g9h3aI1W
PI6dcAvFIS6cVbCMX5sOKLv1Av1h1cXXsilk4
3oZpVIQ137PrJSD6VftZib1I3GDEKWAfcg5naTtu
oQipJ71-
RTI60MTyGqMIE19M00BIfWavh2PtFc8NMCYNCo
hYy-M50P4DrnbC_jEQSx_2Q-
_NmEjtzkETUzLz3z12Bbgnp7XM-
Supepgs405COT8Kj5-
PWw_p1xaoxeDmJVPJs4o48tTvfVxxWceBT6Y6Q
TlzoFaGXGEK1aIFetLOTHVhnXAb9gRiCvvWRjt
u4cBFYapUKFoe7ozSkpZVLz0cuJbQwF1cwaKt
t0nnR4fX-
TWDHvY_zgjkLzaPSJh0TmyCs6P20tk6Mx0eH
wN1vTdJ8evUprfgM1EzWf134td1J3_dHFNLCZI
EXbA1L1iGaxEhQmH880Z9y6XTmP4N4XHP1Tvg
DHQPhn1yLeLJMtU1dcIAF19fkskv02mV15iUu
9MK1Ai5nm8WWSJRhQq1G1Woi_uNrc0D1UGE_0
Dm80JQ8LzIHNTG6CSMFC9HdQIBBH

```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```

{
  "typ": "JWT",
  "alg": "RS256"
}

```

PAYLOAD: DATA

```

{
  "iss": "OAuth Server",
  "sub": "OID:",
  "4zQmVxZryPRXNhpMrbdEmjTdcHyEgkUbBnv3Ebx3TDhpK2",
  "iat": 1723244836,
  "exp": 1723244836,
  "cnf": {
    "jwk": {
      "kty": "RSA",
      "n": "wOhA1a9n9KbyhV9ZXN62c9E3QrnCPeG3NcSpS8p-
ZT-
YtqvsgyP1SBtGN8TPEjwsg0BfAdsdAGuIm63GmZeFqRA748njWYwUq
FOS3nRAndvRcFvRovocCjh43Khd1JyYpXfu8YuCapMgBcURetiwwPK
ycnVZ7vCPBRPeymNf6Gkg0rwa2K1FFKE-
CttgrPeqvLPhndycwrHG10T8_MgEPxmoXr6wBrH1Dz4nLZfmVfc6UK
BmIRZ7JgcJMPETMqetUmAd_9Vwwq9nnDhgDz8e7GIBMqZSw6YkSAJV
nB13-FA1uXCFby1K8IRvjVtY7987ImOFVUMD4EgyV8Qw",
      "e": "AQAB"
    }
  },
  "access_token":
    "8a782b398bbcc39b796fcd82bd84c206153c69bd",
  "resource_server": "ResourceServer: Printing Server"
}

```

VERIFY SIGNATURE

```

RSASHA256(
  base64urlEncode(header) + "." +
  base64urlEncode(payload),
  Public Key in SPKI, PKCS #1,
  X.509 Certificate, or JWK string
  format.

  Private Key in PKCS #8, PKCS #1,
  or JWK string format. The key
  never leaves your browser.
)

```

Invalid Signature

SHARE JWT

- To resolve this, copy the contents of “*MyServer.crt*”, which is the public key of AS. Paste it in the “*Verify Signature*” window. It will automatically validate the signature of the token and you get “*Signature Verified*” message.

The screenshot shows a JWT decoder interface. On the left, under 'Encoded', is a long base64-encoded string. On the right, under 'Decoded', the token is broken down into its header, payload, and signature. The header specifies 'typ': 'JWT' and 'alg': 'RS256'. The payload contains standard JWT claims like 'iss', 'sub', 'iat', 'exp', 'nbf', and 'jwk'. The signature is a long base64 string. At the bottom left, a red box highlights the 'Signature Verified' status. At the bottom right, there is a 'SHARE JWT' button.

5.2 Replay Attacks

- The goal is to ensure that intercepted PoP tokens cannot be reused by an attacker.
- From Postman send the request with PoP token to RS. For the first time, if the token is valid, you’ll get “*Access granted*” as response.
- Click on send again from Postman to send the request to RS with the same token.
- For the second time, you’ll get “*Token already used*” error. Thus, a token can already be used only once avoiding replay attacks.

No.	Time	Source	Destination	Protocol	Length	Info
199	53.022846	76.223.31.44	192.168.1.7	TCP	66	[TCP Keep-Alive ACK] 443 → 57827 [ACK] Seq=1 Ack=2 Win=116 Len=0 SLE=1 SRE=2
202	54.785014	192.168.1.7	104.18.31.2	TCP	54	60697 → 443 [FIN, ACK] Seq=8331 Ack=2618 Win=130816 Len=0
203	54.804460	104.18.31.2	192.168.1.7	TCP	60	443 → 60697 [FIN, ACK] Seq=2618 Ack=8332 Win=65536 Len=0
204	54.804530	192.168.1.7	104.18.31.2	TCP	54	60697 → 443 [ACK] Seq=8332 Ack=2619 Win=130816 Len=0
205	54.987111	192.168.1.7	162.247.241.14	TLSv1.3	1154	Application Data
206	55.009916	162.247.241.14	192.168.1.7	TCP	60	443 → 60692 [ACK] Seq=2281 Ack=6117 Win=24576 Len=0
207	55.165209	162.247.241.14	192.168.1.7	TLSv1.3	457	Application Data
208	55.208354	192.168.1.7	162.247.241.14	TCP	54	60692 → 443 [ACK] Seq=6117 Ack=2684 Win=130816 Len=0
209	56.061476	192.168.1.7	52.111.236.19	TCP	1494	59495 → 443 [ACK] Seq=57 Ack=1 Win=1023 Len=1440 [TCP segment of a reassembled PDU]
210	56.061476	192.168.1.7	52.111.236.19	TLSv1.2	829	Application Data
211	56.070893	52.111.236.19	192.168.1.7	TCP	54	443 → 59495 [ACK] Seq=1 Ack=2272 Win=16386 Len=0
212	56.071823	52.111.236.19	192.168.1.7	TLSv1.2	192	Application Data
213	56.071865	192.168.1.7	52.111.236.19	TCP	54	59495 → 443 [ACK] Seq=2272 Ack=139 Win=1023 Len=0

> Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device
 > Ethernet II, Src: AzureWaveTec_cc:b0:46 (1c:ce:51:cc:b0:46), Dst: SernetTechno_7d:79:6
 > Internet Protocol Version 4, Src: 192.168.1.7, Dst: 162.247.241.14
 > Transmission Control Protocol, Src Port: 60689, Dst Port: 443, Seq: 1, Ack: 1, Len: 0

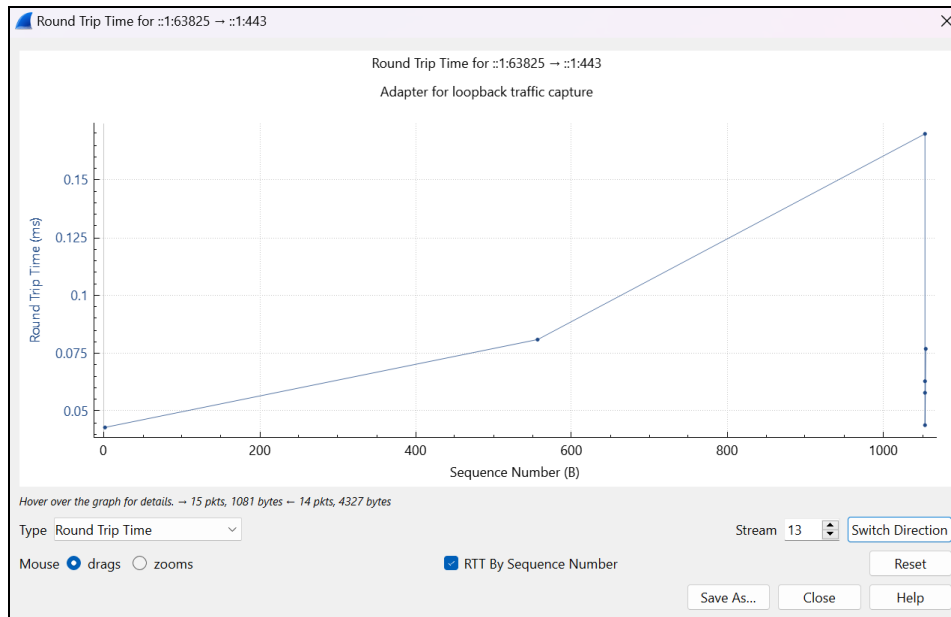
- Send the POST request from Postman to the AS to get the PoP access token. This request will be captured by Wireshark.
- Go back to Wireshark and click the red "Stop" button to stop capturing traffic. Identify your HTTP request.

No.	Time	Source	Destination	Protocol	Length	Info
57	0.189436	:::1	:::1	TCP	76	443 → 63825 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
58	0.189585	:::1	:::1	TCP	64	63825 → 443 [ACK] Seq=1 Ack=1 Win=327168 Len=0
59	0.189779	:::1	:::1	TCP	64	63824 → 443 [FIN, ACK] Seq=1 Ack=1 Win=327168 Len=0
60	0.190823	:::1	:::1	TCP	64	443 → 63824 [ACK] Seq=1 Ack=1 Win=2160640 Len=0
61	0.190813	:::1	:::1	TLSv1.3	619	Client Hello (SNI=localhost)
62	0.190830	:::1	:::1	TCP	64	443 → 63825 [ACK] Seq=1 Ack=1 Win=2159872 Len=0
63	0.191344	:::1	:::1	TCP	64	443 → 63824 [FIN, ACK] Seq=1 Ack=2 Win=2160640 Len=0
64	0.191403	:::1	:::1	TCP	64	63824 → 443 [ACK] Seq=2 Ack=2 Win=327168 Len=0
65	0.197575	:::1	:::1	TLSv1.3	309	Server Hello, Change Cipher Spec, Application Data, Application Data
66	0.197082	:::1	:::1	TCP	64	63825 → 443 [ACK] Seq=220 Ack=240 Win=320912 Len=0
67	0.199233	:::1	:::1	TLSv1.3	562	Change Cipher Spec, Application Data, Application Data

- Note down the timestamp for “Client Hello” and “Server Hello” calls. Calculate the RTT like below:

$$\begin{aligned}
 RTT &= \text{Server Hello Timestamp} - \text{Client Hello Timestamp} \\
 RTT &= 0.005631 - 0.002143 = 0.003488 \text{ seconds} \\
 RTT &= 3.488 \text{ milliseconds}
 \end{aligned}$$

- Thus, RTT for the presented solution is **3.488ms**.
- Additionally, you can generate the RTT graph in Wireshark. Right click on “**Client Hello**” → Follow → TCP Steam → Click on Statistics tab → TCP Steam graphs → Round Trip Time.



7 Common Errors

7.1 Apache SSL certificate error

- Ensure that the location to the “**MyServer.crt**” and “**MyServer.key**” in “**httpd-ssl.cnf**” file for apache is correct.

```
SSLCertificateKeyFile "${SRVROOT}/conf/server.key"
SSLCertificateFile "${SRVROOT}/conf/server.crt"
```

7.2 Invalid Signature even after adding MyServer.crt

- Ensure to copy the entire contents of “**MyServer.crt**”, including “**BEGIN CERTIFICATE**” and “**END CERTIFICATE**”.

7.3 Client denied by server configuration error

- If you see below error in “**errors.log**” file then, check your CN name while certificate CSR generation. “**CN=localhost**” to run this setup.

```
[Fri Aug 02 11:41:40.204281 2024] [authz_core:error] [pid 17132:tid 1092] [client ::1:61963] AH01630: client denied by server configuration:
C:/Users/ketki.jakatdar/Desktop/Thesis/Code/Oauth/oauth2-server-1/oauth2-server-php/favicon.ico, referer: https://localhost/
[Fri Aug 02 11:41:42.301963 2024] [authz_core:error] [pid 17132:tid 1092] [client ::1:61963] AH01630: client denied by server configuration:
C:/Users/ketki.jakatdar/Desktop/Thesis/Code/Oauth/oauth2-server-1/oauth2-server-php/
[Fri Aug 02 11:41:42.322376 2024] [authz_core:error] [pid 17132:tid 1092] [client ::1:61963] AH01630: client denied by server configuration:
C:/Users/ketki.jakatdar/Desktop/Thesis/Code/Oauth/oauth2-server-1/oauth2-server-php/favicon.ico, referer: https://localhost/
```

```
Country Name (2 letter code) [AU]:IR
State or Province Name (full name) [Some-State]:Dublin
Locality Name (eg, city) []:Dublin
Organization Name (eg, company) [Internet Widgits Pty Ltd]:NCI
Organizational Unit Name (eg, section) []:cybersecurity
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:bengette142610@gmail.com
```