Herausgeber et al. (Hrsg.): SKILL 2018,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 11

OIDC-Agent: Managing OpenID Connect Tokens on the Command Line

Gabriel Zachmann [1,2]

Abstract: OpenID Connect is widely used in Authentication and Authorization Infrastructures including the infrastructures of multiple EU projects like INDIGO-DataCloud, the Human Brain Project or the European Open Science Cloud. Due to their nature, OpenID Connect Access Tokens are currently not straightforward to use from the command line. They have a high character count and are short lived. Therefore, they de facto have to be copied from a source providing the access token, most likely a web service. Considering this insufficient usability from the command line, our goal was to overcome this by developing a tool to manage OpenID Connect tokens. We present the design of this tool named `oidc-agent` and possible usages. The design is oriented at the `ssh-agent`, providing the user a familiar way to handle OpenID Connect tokens. By splitting the whole service into multiple components we also ensure privilege separation. We implemented a daemon to manage OpenID Connect tokens (`oidc-agent`), a tool for generating agent account configurations (`oidc-gen`) and a tool for loading and unloading these configurations from the agent (`oidc-add`). Additionally, we provide application programming interfaces for agent clients through C and UNIX domain sockets. We also provide an example agent client (`oidc-token`) that can be used to easily get an access token from `oidc-agent` using the command line. Therefore, users do not need to handle long, unhandy access tokens, but the application can obtain an access-token through `oidc-agent` when needed. All components can be freely used and are available on GitHub[3] under the MIT license.

Keywords: OpenID Connect; OIDC; oidc-agent; authorization; authentication; security; command line

1 Introduction

OpenID Connect (OIDC) [Sa14] is an authentication protocol based on OAuth2 [Ha12] and an important key component in many modern Authentication and Authorization Infrastructures (AAIs) including those of several EU projects. At the Karlsruhe Institute of Technology (KIT) we are engaged in multiple of these projects in the field of AAI. Users and developers of these projects and other AAIs have to handle OIDC access tokens on a regular base. While for web applications this is easy to do, handling OIDC access tokens on the command line is currently a cumbersome and repetitive procedure requiring manual copying and pasting.

[1] Karlsruhe Institute of Technology, Steinbuch Centre for Computing, Herrmann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany gzachmann@outlook.com
[2] Baden-Wuerttemberg Cooperative State University Karlsruhe, Germany
[3] https://github.com/indigo-dc/oidc-agent

Currently there is no other tool for managing OIDC tokens on the command line. However, there are command line tools for dynamically registering OIDC clients [Go15; Sk17], but these tools are only for registering and managing clients and not for handling access tokens. Therefore, our goal was to develop an architecture for a tool that can do both: registering clients and handling OIDC tokens. By implementing the developed architecture we wanted to integrate OIDC with the command line with just one tool chain.

2 Architecture

While developing the architecture of `oidc-agent` [IN18] we followed the `ssh-agent` [Op17] design, because `oidc-agent` has a similar purpose as the `ssh-agent`, but for OIDC tokens instead of ssh keys. By following the `ssh-agent` design, users are able to use `oidc-agent` in a way they are used to from `ssh-agent`.

The architecture consists of multiple components:

`oidc-agent`: The actual agent managing the tokens and performing all communication with the OpenID Provider (OP).

`oidc-gen`: A tool for generating account configuration files for usage with `oidc-agent` and `oidc-add`.

`oidc-add`: A tool that loads the account configurations into `oidc-agent`.

`oidc-token` and third party applications: Applications that need an OIDC access token can obtain it through the agent's application programming interface (API). One example application for getting an access token is `oidc-token`.

We will describe these components in the following subsections. The architecture of `oidc-agent` is visualized in Figure 1.

2.1 `oidc-agent`

`oidc-agent` is the central component of the `oidc-agent` project. It runs as a daemon in the background and handles all communication with the OPs. Other applications (including `oidc-gen`, `oidc-add`, and `oidc-token`) have to use an UNIX Domain Socket to communicate with the agent. To locate the socket an environment variable is used. This variable has to be set when starting the `oidc-agent`. The required shell command is printed by `oidc-agent` on startup. This is the same process as it is for `ssh-agent` [Ma16]. The access control for the used socket is handled by the file system. Therefore, any process started by the same user can communicate with the agent.

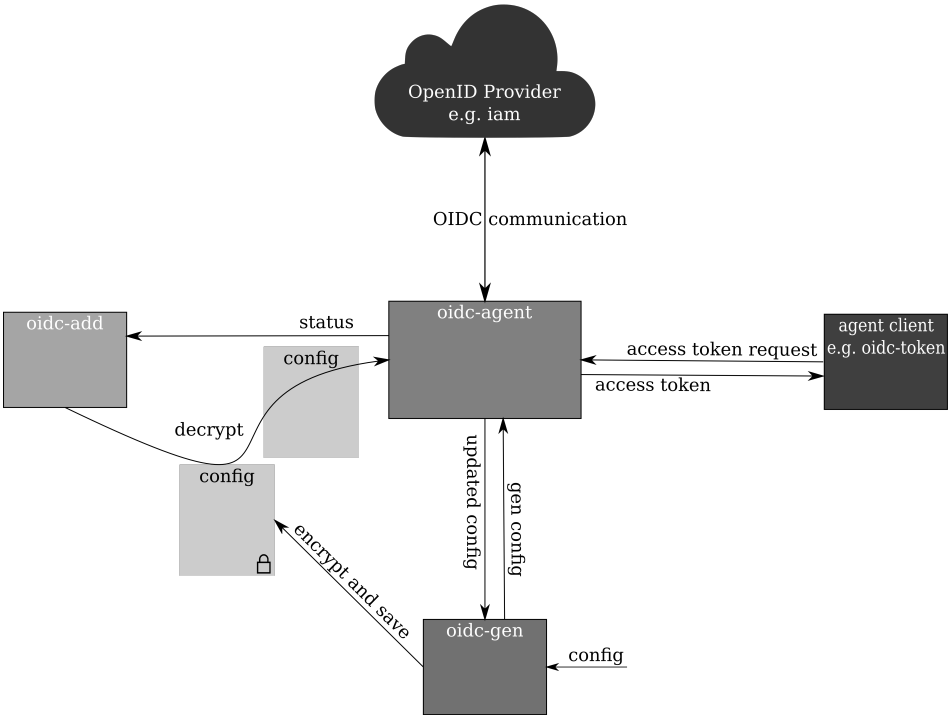

Fig. 1: Architectural Design of `oidc-agent`

2.2 `oidc-gen`

`oidc-gen` is used to generate account configuration files. To use `oidc-agent` with an OP an account configuration is needed. To be able to obtain access tokens `oidc-agent` needs a registered OIDC client. Some OPs support dynamic client registration which can be used by a client to register itself with the OP. For these providers dynamic client registration can be used by `oidc-gen` and `oidc-agent` to register the client and then generate the account configuration file. If the OP does not support dynamic client registration, users have to register a client themselves and then provide the relevant information to `oidc-gen` to create the configuration file.

2.3 `oidc-add`

`oidc-add` is used to add an account configuration to the agent. `oidc-add` will read the encrypted account configuration file, decrypt it with the user supplied password and send the configuration to `oidc-agent`. After adding a configuration an access token can be obtained

for this configuration (e.g. by using `oidc-token`). `oidc-add` can also be used to remove an already loaded configuration from the agent.

2.4 `oidc-token`

`oidc-token` is an example application that is able to obtain an access token through `oidc-agent`. Other applications that need to get an access token can use the provided API. There are two ways of using the API. We provide a C-API that hides the communication details under simple function calls. Therefore, other applications do not have to handle locating, writing and reading of the UNIX domain socket. Many languages support calling C functions which makes it easy to integrate in any application. The C-API can be used by including the source files or by using the provided library. `oidc-token` uses this C-API and can be used as a reference implementation on how to use this API.

If an application cannot or does not want to call C functions, it can communicate directly with `oidc-agent` through the UNIX Domain Socket. To do so, the application has to obtain the socket path from the environment and connect to it. Socket communication is done through JavaScript Object Notation (JSON) encoded messages.

2.5 Privilege Separation

By following the security by design principles and splitting the system's functionalities into multiple components we also achieved privilege separation. Table 1 shows the privileges every component needs. We emphasize that the only file `oidc-agent` reads is the Certificate Authority (CA) bundle file needed for Transport Layer Security (TLS) encrypted communication. So the agent - as the only component that has network access - does not access disk (with the CA bundle file as an exception). Please also note that `oidc-gen` does not need to execute any files to work correctly. However, it needs the execution right for automatically opening the authorization uniform resource locater (URL) in a web browser when performing the Authorization Code Flow.

component	ipc	network	read file	write file	execute file
`oidc-agent`	✓	✓	(✓)	✗	✗
`oidc-gen`	✓	✗	✓	✓	(✓)
`oidc-add`	✓	✗	✓	✗	✗
agent clients	✓	✗	✗	✗	✗

Tab. 1: Privilege Separation in the oidc-agent project

2.6 Usage of OpenID Connect Flows

As already mentioned, `oidc-agent` is able to use dynamic client registration to automatically register an OIDC client, but `oidc-agent` supports more OIDC flows. Normally, the agent uses the Refresh Flow to obtain an access token using a refresh token. This flow is illustrated in Figure 2. The used refresh token is stored in an encrypted way and is obtained when the account configuration is generated. `oidc-agent` supports multiple ways to obtain this refresh token.

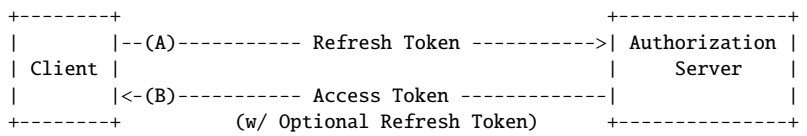

Fig. 2: Refreshing an Expired Access Token. Adapted from [Ha12]

- **Out of Band**: The refresh token can be obtained out of band and be provided directly. But it is very unlikely that a user is able to obtain a refresh token for the used client, because refresh tokens are bound to a specific client id. This functionality is included for development and for potential advanced use cases.
- **Password Flow**: The Resource Owner Password Credentials can be used directly as an authorization to obtain a refresh (and access) token [Ha12]. Obviously this flow reveals the user's credentials to `oidc-agent`. We emphasize that `oidc-agent` does not store this information and that it is held in memory as short as possible, but users might want to use one of the other flows that do not reveal the user's credentials to `oidc-agent`. However, this is the only flow that can be done entirely on the command line; the other flows require some sort of web-based authentication. This flow is visualized in Figure 3.
- **Authorization Code Flow**: The Authorization Code Flow is the most widely spread OAuth2 / OIDC flow and is the standard web flow. No user credentials are revealed to `oidc-agent`, instead the user authenticates against the OP using a web browser. This flow requires `oidc-agent` to start a small web server to receive the OP's response. Out of the implemented flows this is the one mostly supported by OPs; thus it is also the default flow for `oidc-agent`. Figure 4 shows how the authorization code flow works.
- **Device Flow**: The Device Flow is an OAuth2 flow specially for browserless and input constrained devices [De17]. It uses a second device to perform the authentication against the OP in a browser and in that way also does not reveal the credentials to `oidc-agent`. Because the authentication is done on a second device, there is no web interaction needed on the primary device, i.e. on the primary device it can be done using only the command line. The flow is illustrated in Figure 5.

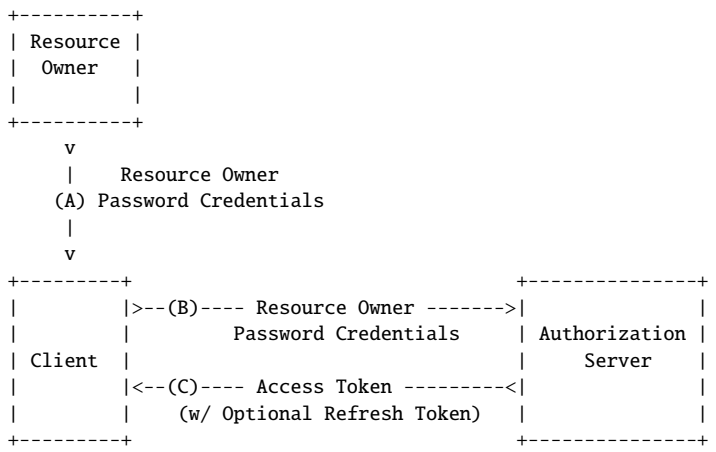

Fig. 3: Resource Owner Password Credentials Flow [Ha12]

The Authorization Code Flow is clearly the most used flow to obtain a refresh token in `oidc-agent`, because it is supported by almost any OP and because the password flow and device flow are supported by only few OPs. Please note that the device flow specification is currently not in the state of being an RFC but only an Internet Engineering Task Force (IETF) draft.

2.7 General Principles

Because `oidc-agent` is a security related software that handles sensitive data, we have to handle it with the necessary caution. As already mentioned, user credentials are not stored on disk and kept in memory as short as possible. To ensure this, we clear all allocated memory before freeing it. By clearing all allocated memory and not only sensitive data we ensure that we do not accidentally leak information (e.g. a server response that contains a refresh token as a sub-string).

The stored data contains sensitive elements - in particular the refresh token and client secret. All data written to disk is therefore encrypted using the easy to use, cross-platform encryption library `libsodium` [17a]. The used encryption algorithm is `XSALSA20`. It is a stream cipher based on `SALSA20` but with a 192 bits long nonce instead of 64 bits [17b]. XSalsa20 uses a 256-bit key, that is derived from the users's encryption password, as well as the first 128 bits of the randomly generated nonce to compute a subkey [17b]. This subkey and the remaining 64 bits of the nonce are the parameters for the `Salsa20` function used to eventually generate the stream [17b].

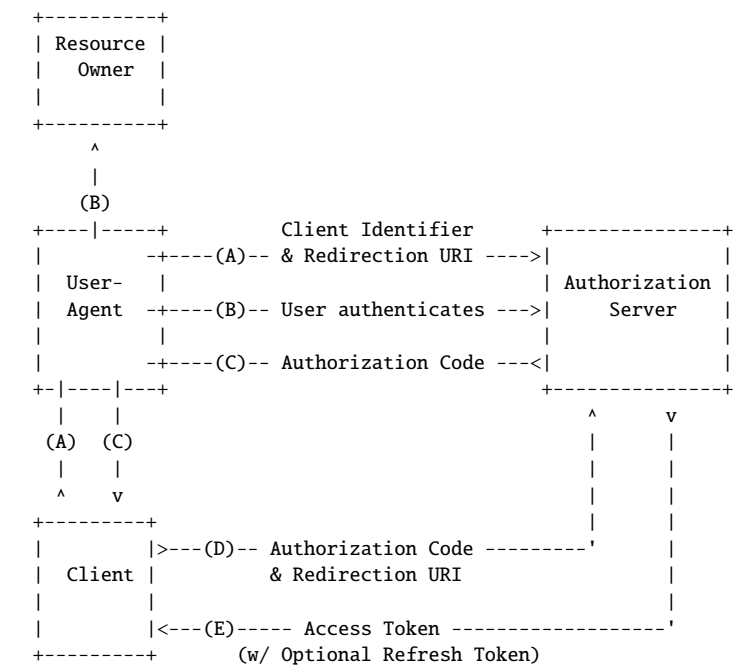

Fig. 4: Authorization Code Flow [Ha12]

3 Use Cases

Usages of `oidc-agent` can be categorized into two groups. An application can utilize `oidc-agent` to obtain an access token or the user can obtain the access token from the agent and provide it to an application. For both of these groups we will describe possible use cases.

3.1 Applications Utilizing `oidc-agent`

Using an application that utilizes `oidc-agent` to obtain an access token is very easy. Instead of manually obtaining an access token and providing it to the application (e.g. through an environment variable) the application has to be modified to support `oidc-agent` natively. The application then only needs the name of the account configuration to be used. In addition, `oidc-agent` has to be running and the account configuration has be loaded, i.e. by using `oidc-add`. Then the application can be used as normal.

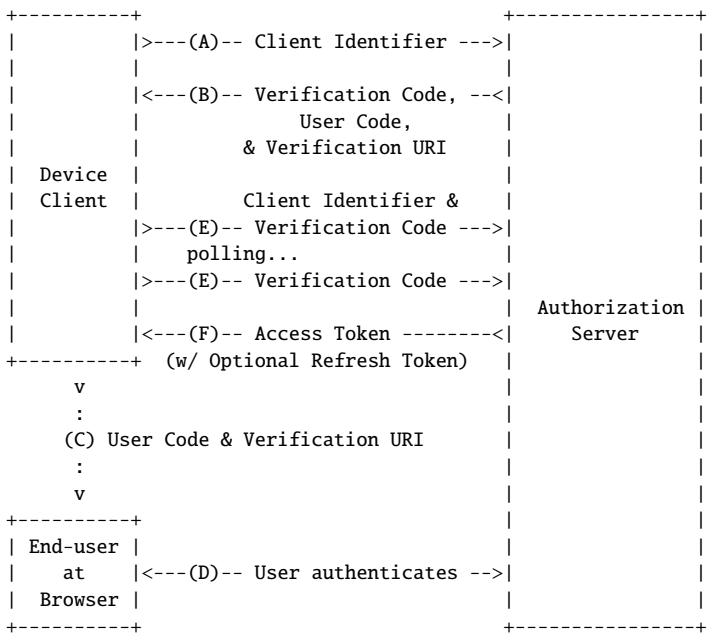

Fig. 5: Device Flow [De17]

An example of an application that can utilize `oidc-agent` to obtain access tokens is `wattson`. `wattson` [IN17b] is a command line client for `WaTTS` [IN17a], the INDIGO [18d] Token Translation Service (TTS) that translates OIDC credentials to non-OIDC credentials (e.g. an X.509 certificate). An example on how to use `wattson` with `oidc-agent` is shown in Listing 1. Lines 1–3 initialize the environment variables needed by `wattson`; lines 4–8 startup `oidc-agent` and load the "indigo-iam" account configuration; line 10–16 show the actual `wattson` call and the resulting output.

3.2 Using `oidc-agent` to Obtain an Access Token

If an application does not use our API, but needs an OIDC access token, `oidc-token` can be used to obtain the access token from `oidc-agent`. Most likely the application will read the access token from an environment variable. Setting this variable with `oidc-token` is very straightforward as can be seen from Listing 2. Again lines 1–6 show the agent startup and the loading of the account configuration. In line 7 the environment variable is filled with the requested access token; after that it can be used by the application.

```
user@oidc:~$ export WATTSON_URL=https://watts.data.kit.edu
user@oidc:~$ export WATTSON_ISSUER=iam
user@oidc:~$ export WATTSON_AGENT_ACCOUNT=indigo-iam
user@oidc:~$ eval `oidc-agent`
Agent pid 2018
user@oidc:~$ oidc-add indigo-iam
Enter encrpytion password for account config indigo-iam:
success

user@oidc:~$ wattson request x509
connecting to https://watts.data.kit.edu/api/v2/iam/ using protocol version 2
received token from oidc-agent
requesting credential for service [x509]:
Credential [a2fa1b39-8723-4385-aae1-6d768a5a2f88]:
[ Certificate (textfile)] => Certificate:
...
```

List. 1: Using `wattson` with `oidc-agent`

```
user@oidc:~$ eval `oidc-agent`
Agent pid 2018
user@oidc:~$ oidc-add indigo-iam
Enter encrpytion password for account config indigo-iam:
success

user@oidc:~$ export WATTSON_TOKEN=`oidc-token indigo-iam`
```

List. 2: Using `oidc-token` to pass an access token through an environment variable

Of course `oidc-token` may as well be used to get the access token and then use it in any other way necessary. To obtain an access token from `oidc-agent` with `oidc-token` a simple call to `oidc-token` providing the account configuration name is enough.

One scenario in which this access token might be used is inside the Human Brain Project (HBP). The HBP [18c] is a Future and Emerging Technologies (FET) flagship project as part of the Horizon 2020 programme [Eu] of the European Union (EU) and aims at a better understanding of the human brain. Users of the HBP can use a High Performance Computing (HPC) system consisting of supercomputing sites in Jülich, Barcelona, Bologna and Lugano and a cloud storage system in Karlsruhe [Be16]. This HPC system can be accessed through the UNiform Interface to COmputing REsources (UNICORE) [18e] by using OIDC. This means that `oidc-agent` can be used to get an access token for accessing the HPC system.

4 Conclusion

With `oidc-agent` we developed a tool for managing OIDC access tokens on the command line. The architecture is based on `ssh-agent` and provides the user a way to handle OIDC tokens similar to ssh keys. The `oidc-agent` architecture also supports privilege separation and secure memory deallocation.

Users can easily generate new account configurations with `oidc-gen`. After the account configuration is loaded into `oidc-agent` using `oidc-add`, an application that needs an access token can request it using the API. Additionally `oidc-token` can be used to get an access token on the command line.

`oidc-agent` supports several OIDC flows like the password flow, authorization code flow and device flow. This allows `oidc-agent` to be widely used with many different OPs. We verified our implementation with B2Access [EU16], EGI-Checkin [18b], Elixir [18a], Google [Go], HBP and INDIGO IAM.

`oidc-agent` is released under the MIT license as part of the INDIGO-Datacloud project. Source code and releases are available at GitHub[4]. To improve the installation process we want to add the oidc-agent package into the repositories of the major linux distributions.

Acknowledgments

This project was done has part of my dual study at the Baden-Wuerttemberg Cooperative State University Karlsruhe within the practical phase at the KIT-SCC.

I want to thank my supervisors Bas Wegh and Marcus Hardt for their feedback, guidence, and advise throughout the project.

Additionally, I want to thank the anonymous reviewers for their comprehensive feedback.

References

[17a] libsodium | A modern, portable, easy to use crypto library, 2017, URL: `https://github.com/jedisct1/libsodium`.

[17b] The Sodium crypto library (libsodium), 1.0.13, libsodium, July 2017, URL: `https://www.gitbook.com/book/jedisct1/libsodium/details`, visited on:

[18a] ELIXIR | A distributed infrastructure for life-science information, 2018, URL: `https://www.elixir-europe.org/`.

[18b] European Grid Infrastructure, 2018, URL: `https://www.egi.eu/`.

[4] https://github.com/indigo-dc/oidc-agent

[18c] Human Brain Project, 2018, URL: https://www.humanbrainproject.eu/.

[18d] INDIGO DataCloud, 2018, URL: https://www.indigo-datacloud.eu/.

[18e] UNICORE | Distributed computing and data resources, 2018, URL: https://www.unicore.eu/.

[Be16] Benedyczak, K.; Schuller, B.; Sayed, M. P.-E.; Rybicki, J.; Grunzke, R.: UNICORE 7 - Middleware Services for Distributed and Federated Computing. In: 2016 International Conference on High Performance Computing Simulation (HPCS). Pp. 613–620, July 2016.

[De17] Denniss, W.; Google; Bradley, J.; Identity, P.; Jones, M.; Microsoft; Tschofenig, H.; Limited, A.: OAuth 2.0 Device Flow for Browserless and Input Constrained Devices, tech. rep., Oct. 2017, URL: https://tools.ietf.org/html/draft-ietf-oauth-device-flow-07.

[Eu] European Commission: Horizon 2020 - European Commission, URL: https://ec.europa.eu/programmes/horizon2020/.

[EU16] EUDAT: B2ACCESS - EUDAT, Nov. 2016, URL: https://www.eudat.eu/services/b2access.

[Go] Google: Google Identity Platform, URL: https://developers.google.com/identity/.

[Go15] Goering, B.: openid-cli: A CLI for interacting with an OpenID Connect provider, 2015, URL: https://github.com/gobengo/oidc-cli,%20visited%20on:%20June%206,%202018.

[Ha12] Hardt, D.: The OAuth 2.0 Authorization Framework, RFC 6749, RFC Editor, Oct. 2012, URL: http://www.rfc-editor.org/info/rfc6749.

[IN17a] INDIGO-Datacloud: WaTTs - the INDIGO Token Translation Service, 2017, URL: https://github.com/indigo-dc/tts.

[IN17b] INDIGO-Datacloud: wattson: the watts command line client, 2017, URL: https://github.com/indigo-dc/wattson.

[IN18] INDIGO-Datacloud: oidc-agent for managing OpenID Connect tokens, 2018, URL: https://github.com/indigo-dc/oidc-agent.

[Ma16] Manual, L. P.: SSH-AGENT(1) BSD General Commands Manual, Nov. 2016, URL: http://man7.org/linux/man-pages/man1/ssh-agent.1.html.

[Op17] OpenBSD: OpenSSH, 2017, URL: https://www.openssh.com/.

[Sa14] Sakimura, N.; Bradley, J.; Jones, M.; de Medeiros, B.; Mortimore, C.: OpenID Connect Core 1.0 incorporating errata set 1, tech. rep., Nov. 2014, URL: http://openid.net/specs/openid-connect-core-1_0.html.

[Sk17] Skokan, F.: openid-client-cli: CLI for managing dynamic OpenID Connect client registrations. 2017, URL: https://github.com/panva/openid-client-cli.