## Identity Provider

### Description of Component

The Identity Provider (IdP) provides a service that autenticates users, provides proofs of past authentication process (called identity assertions), and checks the validity of such assertions. It also provides IdP-Proxy to the IdModule deployed on the runtime so that this IdModule may interact with the IdP whatever protocol it may use.

Two OpenID Connect-compliant Identity Providers have been customised and deployed for demonstration purpose: 1. the first one is an OAuth2/OIDC server with added support for IdP-Proxy, adapted from [agmoyano/OpenIDConnect](Illustration%20of%20a%20single%20httperf%20connection); 2. the second one is adapted from [Nat Sakumura phpOIDC implementation](https://bitbucket.org/PEOFIAMP/phpoidc).

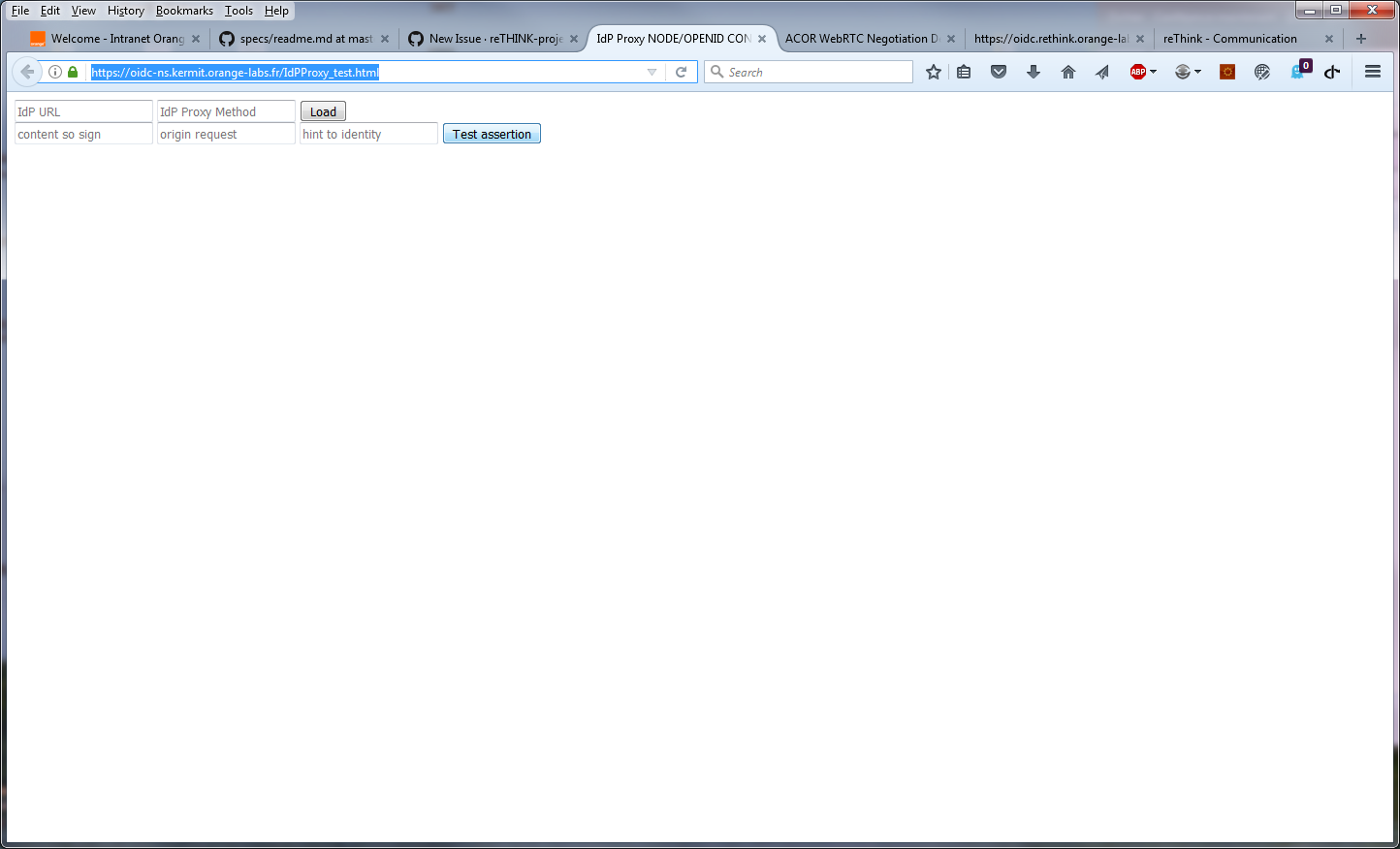
### Metrics

Since IdP is a purely request-based service, the most relevant KPIs would characterize response time, for instance average response time and 9th decile: - for authentication requests, - for assertion generation, - for assertion verification.

However, IdP requests per user should be scarce compared to communication-related exchanges, especially on our testbeds which constitute more of a prototype environment. Moreover, the goal of reTHINK is to let the user choose his/her identity provider, which could thus be Google or Facebook. Therefore, tests limites tho the two provided IdP would not be relevant. Finally, it is not the responsibility of reTHINK to test reference implementation of identity protocols such as OpenID Connect. Indeed, IdPs are peripheral support services and will not be critical in evaluating overall environment performance. As a matter of fact, they are pre-existing (almost 'legacy') platforms whose functionalities are needed within reTHINK project but not results of reTHINK.

### Performance tests

No server performance tests are planned, as IdP servers are Open source project that have been modified to add the IdPProxy feature. This IdP proxy is executed on client side.

The application IdP-Proxy test is installed with the IdP Server and processes 3 actions: it loads the IdP-Proxy if implemented, generates and validates an Identity assertion if the user is logged on this IdP. An example of the application is accessible at https://auth.rethink2.orange-labs.fr/IdPProxy\_test.html. Test application for the IdP Proxy:  


Tests conducted on IdP-Proxy have been conducted using a Firefox platform, and have provided the following results: - Javascript IdP-Proxy loading: - min time : 4ms - max time : 13ms - av time : 8.9ms - Javascript generate assertion - min time : 327ms - max time : 1235ms - av time : 553ms - Javascript validate assertion - min time : 147ms - max time : 296ms - av time : 214ms

It should be noted that the results are highly related to the client machine. Also, the length of the salt has a very low influence on the processing time.

### Identity parameters negotiation tests.

Conformance tests have been conducted to enable the use of IdP-Proxy according to the IETF specifications of the WebRTC Security architecture [3].  
As described in [Corre et al. paper](https://link.springer.com/chapter/10.1007%2F978-3-319-60131-1_27) we evaluated the possibility to deploy negotiation over ACR and IdP's origin with current WebRTC specifications. Our conclusion shows that it is not possible to request ACR to IdP Proxy when calling the generateAssertion function. As a result, the specication would need to be updated to support ACR negotiation. In particular: the generateAssertion function could be extended to accept additional parameters, and the IdentityValidationResult could be left open to extensions.  
We implemented our identity parameters negotiation solution on the CS side as it was the simplest solution. As identity negotiation is most useful in scenarios of inter-operable communication services, such services could be acting as the identity recommendation source. This may seems to contradict the WebRTC trust model with untrusted CS. However, in the interoperable scenario, we may want to relax the trust model and consider that a CS may be trusted by its own user. In this situation a CS could be well-suited to provide recommendation and evaluation of the other-peer's authentication.  
This has been implemented in a simple WebRTC application that uses our IdPProxy.  
Step one:  
Bob is connecting to his WebRTC Application (https://acor-webrtc.rethink2.orange-labs.fr/) using his identity, and opens a room.  
Step two:  
Alice connects to the same application and join bob's room. At this stage, they can communicate, but no identity proof have been exchanged.  
Step three:  
Alice claims to Bob an Identity assertion, with a trust level to 2, and a hint to get a subset of accepted IdPs domains.  
Stet four:  
Bob is receiving the claim and sends his identity assertion.