OLYMPUS PROJECT

OBLIVIOUS IDENTITY MANAGEMENT FOR PRIVATE USER-FRIENDLY SERVICES

D3.3 OLYMPUS Blueprint

PROJECT NUMBER	PROJECT ACRONYM
786725	OLYMPUS
CONTACT	**WEBSITE**
contact@olympus-project.eu	http://www.olympus-project.eu/

Due date of deliverable: 31-08-2020
Actual submission date: 31-08-2020

Dissemination Level	
PU = Public, fully open, e.g. web	✓
CO = Confidential, restricted under conditions set out in Model Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC.	
Int = Internal Working Document	

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

REVISION HISTORY

The following table describes the main changes done in the document created.

Revision	Date	Description	Author (Organization)
V0.1	04/09/2019	ToC and initial content	Anja Lehmann
V0.2	24/09/2019	Comments added	Rafa Torres, Jesus Garcia, Jorge Bernal, Antonio Skarmeta
V0.3	31/03/2020	Added content and details	Julia Hesse, Tore Frederiksen, Michael Stausholm
V0.4	10/06/2020	Comments addressed and content added	Jesus Garcia, Rafa Torres, Jorge Bernal, Antonio Skarmeta
V0.5	05/08/2020	Updated details in relation to the PESTO implementation	Michael Stausholm, Tore Frederiksen
V0.6	27/08/2020	Review comments	Jesus Garcia, Rafa Torres, Jorge Bernal, Antonio Skarmeta, Georgia Sourla
V0.7	27/08/2020	Final version	Michael Stausholm, Tore Frederiksen

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

OBLIVIOUS IDENTITY MANAGEMENT FOR PRIVATE AND USER-FRIENDLY SERVICES

ABSTRACT

This deliverable outlines the overall OLYMPUS framework, including its components and interfaces. Further focus is on the required components to realize the novel cryptographic approaches of the intended ecosystem.

KEYWORDS

Identity Management, framework, deployment, identity provisioning

AUTHORS (ORGANIZATION)

Anja Lehmann (IBM), Julia Hesse (IBM), Patrick Towa (IBM), Michael Stausholm (ALX), Tore Frederiksen (ALX), Antonio Skarmeta (UMU), Rafa Torres (UMU).

DISCLAIMER

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725, but this document only reflects the consortium's view. The European Commission is not responsible for any use that may be made of the information it contains.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

Contents

List of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

1. Introduction

This document provides a blueprint of the OLYMPUS architecture. Besides detailing the core components of the oblivious identity management system developed within the project, it details the functionality of the system. Further, it defines interfaces provided by each component. The architecture is then argued to fulfil the system's general and security requirements described in D3.1. One goal of the architecture is to make the existence of many virtual identity providers (IdPs) as oblivious to the user as possible and preserve a user's view as he would use a system with only one trusted IdP.

The document primarily covers the architecture and certain details are omitted. Further details on the implementation of the system are described in deliverables D4.1, D4.2 and D5.2. As the development of the system is still ongoing, future deliverables D4.3 and D5.4 may also document potential deviation from the descriptions provided in this document. As the OLYMPUS system is quite comprehensive, certain components are critical for the functionality of the system, whereas other components are, while nice to have, not critical for functionality. This means that some features are described theoretically in this deliverable, while they are not included in the reference implementation described in D4.2, D4.3, D5.2 and D5.4.

The critical components include the OPRF functionality for password verification, threshold signatures for token generation, distributed p-ABC issuance and epoch key refreshing. In addition to this core functionality, abstract functionality used in specific deployment, i.e. identity proving and data persistence, is also well described.

The nice to have features not implemented in the reference implementation include hiding user attributes from the IdPs, multifactor authentication and support for predicates in the dp-ABC scheme. While the features are not implemented, the system is implemented such that it is compatible with the features. This allows future development to somewhat easily include these features. Indeed work is being donein order to implement proof-of-concepts of these features.

1.1 Components of the OLYMPUS system

The OLYMPUS system features three main roles: the client, the virtual identity provider and the service provider. These core components of the system are depicted in Figure 1.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

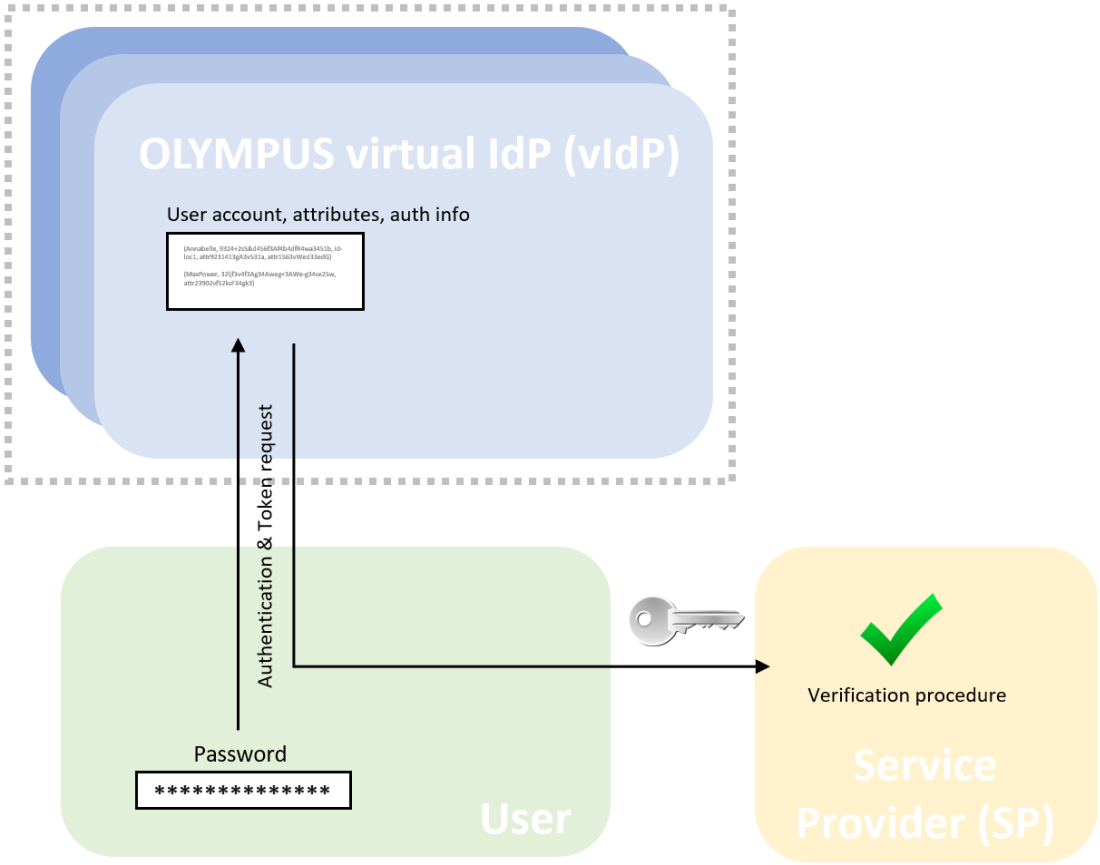


Figure 1: Core components and the core process of the OLYMPUS system. The dotted box indicates that the role of the IdP is fulfilled by multiple virtual IdPs. The process shown consists of several sub-processes, e.g., a user registering an account at the (virtual) IdP(s), a user authentication to the (virtual) IdP(s) and the (virtual) IdP(s) issuing an access token.

2. Functionality

In this section we describe the functionality that the core OLYMPUS components have to provide. As setup, we assume channels of different security levels among the components. In more detail, the following channels are assumed to exist. See Figure 2 for a graphical depiction.

- **Secure channel with server-sided authentication towards users.** An adversary should not be able to mount a man-in-the-middle attack in an ongoing session between a user and a server. Further, the user reliably knows that she communicates with one

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

of the vIdPs. This channel can be implemented using standard methods for securing channels such as TLS and certificate transparency or certificate pinning.

- **Authenticated broadcast channel among servers with guaranteed delivery.** An adversary should not be able to impersonate an IdP when talking to another IdP, which can be handled using mutually authenticated TLS connections with certificates verified using certificate pinning. Ideally, the adversary is not allowed to block any of the messages sent between the IdPs as this presents a denial of service. If guaranteed delivery cannot be achieved, the system will be vulnerable to denial of service attacks. In particular for user registration/password change and refreshing of server keys (which is where inter IdP communication is needed).

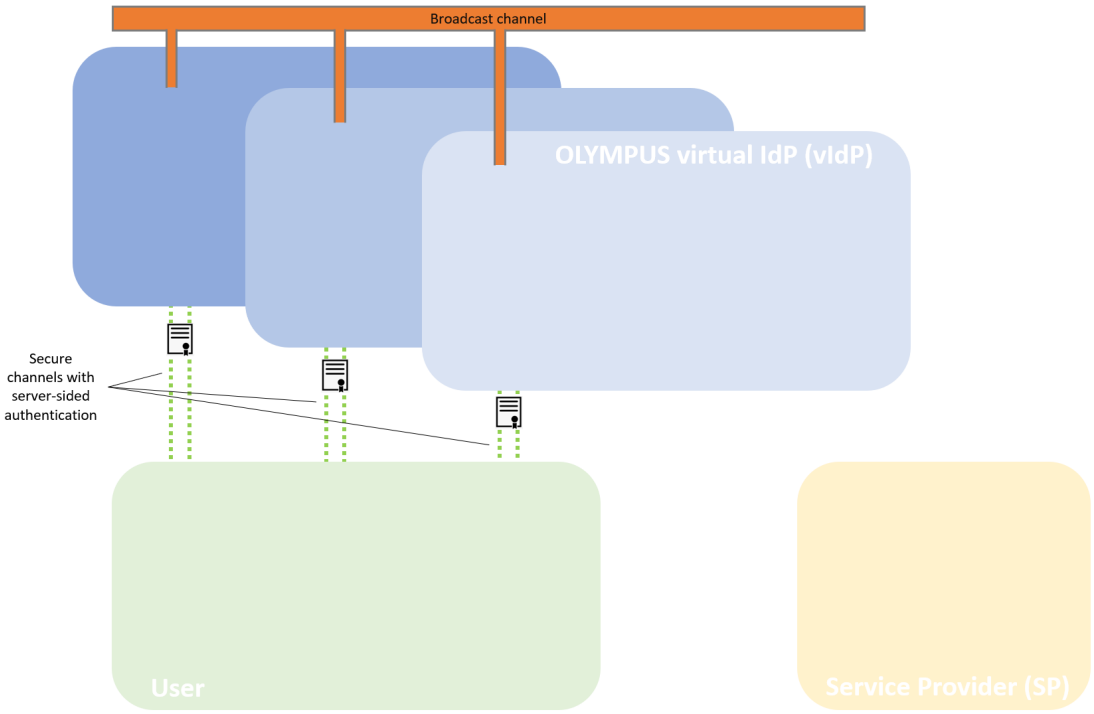


Figure 2: Channel model of OLYMPUS core components.

From a usability point of view, the fact that a *distributed* IdP is used should not be noticeable to the user. This applies both to registration, authentication and account management which we present in the sequel. We discuss possible approaches to virtualize this distributed environment towards the user in Section 3.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

2.1 Setup

The setup phase generates all public and secret key information needed for the distributed identity providers $\mathcal{I}_1, \ldots, \mathcal{I}_n$. For simplicity, we assume a trusted setup phase, where a trusted dealer $\mathcal{D}$ generates and distributes all keys to each of the partial IdPs[1].

$\mathcal{D}.\mathsf{KGen}(param, \mathcal{I}_1, \ldots, \mathcal{I}_n) \rightarrow (pk, sk_1, ms_i, \ldots, sk_n, ms_n)$. A trusted dealer $\mathcal{D}$ on input some public parameters $param$ generates some master secret material ms_i for all identity providers as well as the joint public key pk.

The public key pk contains the identities $\mathcal{I}_1, \ldots, \mathcal{I}_n$ of all identity providers and gets registered with a certificate authority. Each master secret ms_i is sent via a secure channel to the respective identity provider $\mathcal{I}_i$ who stores ms_i in a secure, offline manner. See Section 2.6 for details.

Concretely, each IdP is configured with the master key material and configuration data, using a JSON based format. Amongst other, the configuration contains the following information:

- sharedKey: Containing a modulus, private key and public exponent.
- blindings: A list of blinding values, shared with the other partial IdPs, i.e. servers 0 and 1 share a blinding, servers 0 and 2 share a blinding, etc.
- secrets: A list of master secrets, shared similar to the blindings, i.e. servers 0 and 1 share a secret, etc.
- serverSetup: A list of urls to the other partial IdPs forming the vIdP.

An example of the JSON setup data may be found below:

```
1 {  "sharedKey": {
2       "modulus": 20914650028718510729422497 4...617,
3       "privateKey": 150211483146472103284531...463,
4       "publicExponent": 65537
5    },
6    "blindings": [
7       null,
```

[1] Approaches exist for secure distributed key generation for the these keys if RSA is used for distributed token generation [8].

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

```
        171796972966946380384215069286910616 01...391,
        945794899885076504754268671274844566 75...563
    ],
    "secrets": [
        null,
        103247622215580030538275797845652691 25...323,
        168278234828164409610865193736395657 16...617
    ],
    "serverSetup": [
        "https://server1.com:9081",
        "https://server2.com:9082"
    ]
}
```

The master key material is used during setup to derive epoch keys, denoted sk_i for IdP i. This key material will remain on the partial IdP server, as this will be used for all cryptographic operations. Whereas the master key material will be stored offline in a safe location and will only be used in case of a malicious corruption of one of the partial IdPs.

The setup phase has an extra step for the configuration of the distributed p-ABC scheme. As the selected scheme uses multi-signatures, there is no need to jointly (or through a trusted dealer) perform key generation. However, it is necessary to specify the set of attributes that will be considered for generating credentials. For example, the following JSON setup message can be used to configure an instantiation that contemplates the attributes named “name” and “dateOfBirth”.

```
{   "attributeNames": [
        "name",
        "dateOfBirth"
    ]
}
```

2.2 Registration

Registration allows a client $\mathcal{U}$ to create a fresh user account with the distributed identity providers $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The account is created for a user name uid and protected through a password pwd.

$$\mathcal{U}.\mathsf{Register}(pk, uid, pwd, \{\mathbf{A}, \emptyset\}) \leftrightarrow \\ \mathcal{I}_i.\mathsf{Register}(pk, sk_i, uid, \{\mathbf{A}, \emptyset\}) : (\mathtt{err}/\mathtt{ok}), (\mathtt{err}/\mathsf{acc}_i[uid])$$

This project has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 786725

We model registration as an interactive protocol between the client $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common inputs are a unique user name uid and, *optionally*, a set of attributes $\mathbf{A}$. The user's private input is his password pwd, and each server's private input is its secret key share sk_i. At the end of the registration protocol the client either receives an error code `err` or `ok`, indicating the successful completion of the account creation. Similarly, each server outputs either an error code `err`, or, upon successful creation, the account information $\mathsf{acc}_i[uid]$. Usually, this account information comprises the tuple $(uid, \{\mathbf{A}, \emptyset\}, \mathsf{st}_i[uid])$, where $\mathsf{st}_i[uid]$ is a server-specific state needed for verification of the user's password upon subsequent authentication and login. Specifically it will include the user's public key pk. The *optional* set of attributes $\mathbf{A}$ can be supplied at registration, but may also be provided after a user account has been created, by using the *Add Attribute* method described in Section 2.4

The registration process is implemented as a two-step process between the client and partial IdP. First an OPRF protocol is performed, allowing the client to deterministically generate a public/private keypair. After the client has obtained this keypair, the private key is used to sign a uid and a nonce. The uid, nonce, public key, the optional set of attributes and signature is sent to the IdP. Assuming the process was executed correctly, each server responds with a partial signature on the public key. The client combines the partial signatures and verifies the result. The concrete description of the interfaces as well as used message formats can be found in Section 2.8.

We note that to achieve the goal of distributed and proactive security, the common information about each user that a partial IdP stores will be signed by all the partial IdPs. Concretely this will be done using their respective key shares sk_i and can be verified by the common vIdP public key pk. This is discussed further in Section 4.2.1.

Security Requirements A crucial security goal of our distributed IdP is that as long as not all IdPs are compromised, the passwords – being the "master keys" of the users – cannot be compromised. This comprises the following design requirements:

- The user must not send her password in clear to the IdPs.
- If at most $n - 1$ IdPs are compromised:
  - **Reg-SReq-1:** None of the messages sent by the users to the IdPs must contain information that allows to offline attack

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

the user password.

- **Reg-SReq-2:** Even the collection of $n-1$ stored account informations $\mathsf{acc}_i[uid]$ does not allow the attacker to offline attack the user password.
- **Reg-SReq-3:** An adversary can be removed from compromised IdPs without requiring all the users to perform a registration/enrolment again.

Identity Provisioning We stress that the registration function does not include identity provisioning or vouching, i.e., how the correctness of the attributes is verified is not part of this process. Further, the attributes are an optional input only, allowing users to create an empty account first, for which they can later provide verified attributes through an out-of-band process. Attribute provisioning can be realized by integration of existing identity providers, eID cards or ad-hoc solutions. The identity provisioning process is further described in Section 3.2 and deliverable D5.3 will also touch this subject to some extent.

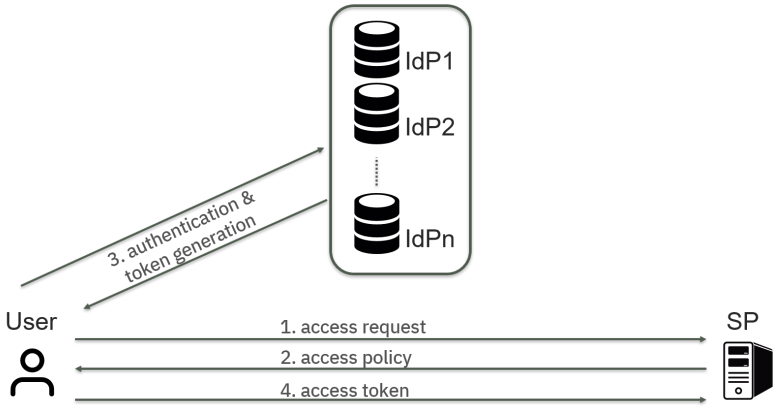
2.3 Authentication

When a user $\mathcal{U}$ wants to access some resource from a service provider $\mathcal{S}$ it first sends an access request message to $\mathcal{S}$, upon which it receives an access policy p. The user then makes an authentication request towards the partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$ with respect to her user name uid and the access policy p, modelled through an interactive protocol $\mathcal{U}.\mathsf{Auth} \leftrightarrow \mathcal{I}_i.\mathsf{Auth}$. Upon successful authentication towards the identity providers, the user receives an authentication token tok which it can forward to the service provider. The service provider uses a verification function Verify to check whether the token tok satisfies the policy p. This flow is quite common, one example of this is the abstract OAuth or OpenId Connect Implicit grant flow, which is further described in Section 3.4.

$\mathcal{U}.\mathsf{Auth}(uid, pwd, p) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Auth}(sk_i, uid, \mathsf{acc}_i[uid], \{p, \perp\}) : (\mathtt{err}/tok), (\mathtt{err}/\mathtt{ok})$.

Authentication and access token generation is an interactive protocol between the user $\mathcal{U}$ and all identity providers $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid and access policy p. The user's private input is her password pwd whereas the private input of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i and user account information $\mathsf{acc}_i[uid]$. Both the user and the partial IdPs will have the vIdP common public key included as

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725



an implicit input as well. At the end of the protocol, the user either receives an error message `err` or an access token tok. The servers end with an error message `err` or the output `ok`, indicating the successful authentication.

If desired, the error code can give an indication why the protocol failed. Possible reasons are a blocked or unknown user account, wrong password, or an access policy that can't be satisfied. We note that some of this information could be regarded as confidential and best practice is to not reveal it, as it could be used to identify user accounts using iteration.

Verify$(pk, p, tok) \rightarrow$ `true`/`false`. The public verification algorithm on input the joint public key pk of the distributed identity provider, a user name uid, access policy p and token tok outputs a verification decision `true` or `false`.

Security Requirements Our distributed IdP must guarantee that 1) the user passwords and password attempts and 2) the joined SSO authentication towards service providers cannot be compromised as long as not all IdPs are corrupted. Thus, any solution implementing these interfaces must ensure the following:

- The user must not send her password attempt in clear to the IdPs.
- If at most $n - 1$ IdPs are compromised:
  - **Auth-SReq-1:** None of the messages sent by the users to the IdPs must contain information that allows to offline attack the user's password attempt.
  - **Auth-SReq-2:** Only a user knowing the correct setup pass-

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

word for uid can successfully authenticate and obtain a valid access token for this user account.

- **Auth-SReq-3:** The IdPs must be able to detect and limit online attacks on a password for a specific uid. That is, the IdPs *must* learn whether a log-in attempt was successful.
- **Auth-SReq-4:** *optional:* The adversary does not learn the access policy p.
- **Auth-SReq-5:** *optional:* The adversary does not learn anything about the access token that is output by the user. In particular when colluding with the service provider SP it cannot link the access token received by SP to a particular authentication session (excluding linkage via identifying attributes and/or timing).
- **Auth-SReq-6:** *optional:* The adversary is not able to link the identity of a user from distinct access tokens received by corrupted, possibly distinct, SPs during different sessions (excluding a potentially included user ID, identifying attributes and/or timing).

Generality for Privacy-Preserving Authentication. The goal of the Olympus project is to devise privacy-preserving solutions for the distributed identity provider. Our interfaces can capture such solutions too. Note that the access policy used in the privacy-preserving solutions must not include the URI of the service provider or other unique identifiers as both would prevent any privacy guarantees.

One possible approach for a fully private generation of access tokens is to let the IdPs issue short-term PABC credentials to the user, who in turn will transform the credential into an unlinkable presentation/access token. Therein, the user part $\mathcal{U}.\mathrm{Auth}$ will first internally assemble the fresh PABC credential (possibly containing all user attributes), and then use the access policy to derive the concrete access token tok that will be presented to the service provider.

Further, we stress that our interfaces do *not* impose a unique user identifier uid. That is, the uid that is used towards a service provider, and is needed to verify the validity of the access token can be different to the uid that the user has chosen towards the identity providers, allowing for pseudonymous authentication. Of course, any solution implementing such a pseudonymous authentication must still ensure the binding between the identity claims and the pseudonym used for

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

authentication.

2.4 Account Management

The user can manage its distributed account with the identity providers, for example allowing her to retrieve the stored identity information, add or delete attributes, deleting the account and doing password maintenance.

Each of these *must* be executed after the user has authenticated towards the vIdP using the following method:

$\mathcal{U}.\mathsf{Login}(uid, pwd) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Login}(sk_i, uid, \mathsf{acc}_i[uid]) : (\mathtt{err}/sk), (\mathtt{err}/\mathtt{ok})$.

Login is an interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user's id uid. The user's private input is her password pwd and the private input of each partial IdP $\mathcal{I}_i$ is his secret key share sk_i and his user account information $\mathsf{acc}_i[uid]$. Both the user and each partial IdP will have the common vIdP public key pk implicitly as part of their input as well. At the end of the protocol, the user either receives an error message $\mathtt{err}$ or a private signing key, sk, linked to her password. Whereas the partial IdPs either receives an error message $\mathtt{err}$ or the output $\mathtt{ok}$, indicating the successful authentication.

Concretely the authentication consists of executing the Distributed Password Verification scheme over TLS as discussed in Section 2.2, i.e. the OPRF protocol, and in the work published as part of OLYMPUS, PESTO [2]. Specifically this proves to the servers that the user actually knows its password and will result in her learning her associated private key sk. The user can then send account management messages to the server signed using the private key through the TLS connection they have already established.

Management of the user-specific information is mainly an application-layer problem, which will be carried out through REST calls. The most essential of these calls can be described as follows:

Retrieve info $\mathcal{U}.\mathsf{Retrieve}(uid, \sigma) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Retrieve}(sk_i, uid, \mathsf{acc}_i[uid]) : (\mathtt{err}/(\mathsf{acc}_i[uid], \mathbf{A})), (\mathtt{err}/\mathtt{ok})$.

Retrieve is a non-interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. Common input is the user name uid and a signature on the message, σ, based on the user's private key sk (learned through $\mathcal{U}.\mathsf{Login}$). The private inputs of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i and the user's cor-

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

responding account information $\mathsf{acc}_i[uid]$ and stored attributes $\mathbf{A}$. Each partial identity provider verifies the user's signature and returns an error message $\mathtt{err}$ if the verification fails or the user does not exist. Otherwise each return $\mathsf{acc}_i[uid]$ and $\mathbf{A}$ and output $\mathtt{ok}$.

Add attribute $\mathcal{U}.\mathsf{Add\text{-}Att}(uid, \mathbf{A}, \sigma) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Add\text{-}Att}(sk_i, uid, \mathsf{acc}_i[uid], \mathbf{A}) : (\mathtt{err}/\mathtt{ok}), (\mathtt{err}/\mathbf{A}, \sigma')$.
Add attributes is a non-interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid, a set of new user attributes, $\mathbf{A}$ and a signature on the message, σ, based on the user's private key (learned through $\mathcal{U}.\mathsf{Login}$). The private inputs of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i, the corresponding user's account information $\mathsf{acc}_i[uid]$ for server i and the new attributes received by the user, $\mathbf{A}$. Each partial identity provider verifies existence of the user, her signature and the validity of the new attributes. If any of these checks fail they return and output an error message $\mathtt{err}$. Otherwise they return $\mathtt{ok}$ and sign $\mathbf{A}$ using their respective key shares sk_i to get a signature σ' and output (and add) $(\mathbf{A}, \sigma')$ to the set of stored user-attributes.

Delete attribute $\mathcal{U}.\mathsf{Delete\text{-}Att}(uid, \mathbf{A}_{id}, \sigma) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Delete\text{-}Att}(sk_i, uid, \mathsf{acc}_i[uid], \mathbf{A}) : (\mathtt{err}/\mathtt{ok}), \mathtt{err}/\mathtt{ok})$.
Delete attributes is a non-interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid, a set of attribute identifiers, $\mathbf{A}_{id}$ and a signature on the message, σ, based on the user's private key (learned through $\mathcal{U}.\mathsf{Login}$). The private inputs of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i, the corresponding user's account information $\mathsf{acc}_i[uid]$ for server i and the attributes associated to the user, $\mathbf{A}$. Each partial identity provider verifies the user's signature and the existence of the user and her attributes to delete. If any of these checks fail they return and output an error message $\mathtt{err}$. Otherwise they return and output $\mathtt{ok}$ and delete the attributes with corresponding attribute IDs from the set of stored user attributes.

Delete account $\mathcal{U}.\mathsf{Delete}(uid, \sigma) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Delete}(sk_i, uid, \mathsf{acc}_i[uid], \mathbf{A}) : (\mathtt{err}/\mathtt{ok}), \mathtt{err}/\mathtt{ok})$.
Delete account is a non-interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid and a signature on the message, σ, based on the user's private key (learned through $\mathcal{U}.\mathsf{Login}$). The private inputs of each

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

partial identity provider $\mathcal{I}_i$ is his secret key share sk_i, along with the user's corresponding account information $\mathsf{acc}_i[uid]$ for server i and the user's attributes $\mathbf{A}$. Each partial identity provider verifies the user's signature and her existence. If any of these checks fail they return and output an error message $\mathtt{err}$. Otherwise they return and output $\mathtt{ok}$ and delete all information stored about the user.

Change password $\mathcal{U}.\mathsf{Change\text{-}Pass}(uid, \sigma) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Change\text{-}Pass}(sk_i, uid, \mathsf{acc}_i[uid]) : (\mathtt{err}/\mathtt{ok}), (\mathtt{err}/\mathsf{acc}_i[uid]')$.
Change password is an interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid and a signature on the message, σ, based on the user's private key (learned through $\mathcal{U}.\mathsf{Login}$). The private inputs of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i, along with the user's corresponding account information $\mathsf{acc}_i[uid]$ for server i. Each partial identity provider verifies the user's signature and her existence. If the check passes then the user and the IdPs execute the Register protocol. If it finished successfully the partial IdPs replace the old user info, acc_i stored that is password specific (including the user's public key) with the result of Register, denoted $\mathsf{acc}_i[uid]'$. The partial IdPs do *not* delete $\mathbf{A}$. Finally they return $\mathtt{ok}$ if everything went well or $\mathtt{err}$ if something went wrong. They output $\mathsf{acc}_i[uid]'$ if everything went well and $\mathtt{err}$ otherwise.

Password reset $\mathcal{U}.\mathsf{Reset\text{-}Pass}(uid, pwd, otp) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Reset\text{-}Pass}(sk_i, uid, \mathsf{acc}_i[uid]) : (\mathtt{err}/\mathtt{ok}), (\mathtt{err}/\mathsf{acc}_i[uid])$.
Password reset is an interactive protocol between the user $\mathcal{U}$ and all identity providers $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid and a password reset key, otp. The user's private input is a new password pwd'. The private inputs of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i, along with the user's stored account information $\mathsf{acc}_i[uid]$ for server i. Prior to execution of the functionality, the user has requested a password reset, which has resulted in the out-of-band generation and distribution of otp. The user and the IdPs execute the Register protocol where the user will use her new password pwd' and use otp as an optional attribute. If the used otp can be verified by the IdP, the partial IdPs replace the old user info, $\mathsf{acc}_i[uid]$ stored that is password specific (including the user's public key) with the result of Register, denoted $\mathsf{acc}_i[uid]'$. The partial IdPs do *not* delete $\mathbf{A}$. Finally they return $\mathtt{ok}$ if everything went well or $\mathtt{err}$ if something went wrong. They output $\mathsf{acc}_i[uid]'$ if everything went well and $\mathtt{err}$ otherwise.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

We note that both the user and each of the partial IdPs will implicitly also have the common vIdP public key pk as part of their input. This will *always* be used by every partial IdP to verify the common data stored on the user; e.g. her attributes $\mathbf{A}$, uid and public key. This is done to ensure that it has not been manipulated at rest. Furthermore, what is signed by the user in each request will implicitly include a description of the method and a time-based session ID as will be discussed in Section 4.

Security Requirements Our approach to account management, based on the underlying authentication mechanism, achieves at least the same as the mandatory security requirements for authentication as described in Section 2.3.

- The user must not send her password attempt in clear to the IdPs.
- If at most $n - 1$ IdPs are compromised:
  - **Man-SReq-1:** None of the messages sent by the users to the IdPs must contain information that allows to offline attack the user's password attempt.
  - **Man-SReq-2:** Only a user knowing the correct setup password for uid can successfully authenticate and gain access to the management functionality for this user account.
  - **Man-SReq-3:** The IdPs must be able to detect and limit online attacks on a password for a specific uid. That is, the IdPs *must* learn whether a authentication attempt was successful.
  - **Man-SReq-4:** Only a user who knows her password *or* the proper password reset key can learn the content of, or make changes to, her account.

2.5 Multi-factor Authentication

Despite the user's password not being stored in brute-forceable form at the partial IdPs there is still a significant risk that a given user succumbs to a phishing attack, a keylogger or does not practice good password hygiene and either reuses her password or simply keeps it in an insecure manner. In case their password gets leaked through such negligence, it is the case that impersonation based purely on password authentication becomes trivial. However, this is not a new prob-

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

lem and several approaches already exist to protect the user in such scenario. In particular One-Time Password (OTP) retrieved through a "second factor" is highly common and can easily be added to the OLYMPUS framework as we discuss below. In general, 2 Factor Authentication (2FA) can be implemented in a number of ways, e.g. using SMS codes or special authenticator devices/applications, hence it is highly deployment-specific which approach is used. In order to support 2FA, the user will need to contact a specific endpoint and perform a 2FA authentication process, out-of-scope of the OLYMPUS project, resulting in a temporary 2FA token. This token will then be used in conjunction with the user's password when the user is logging-in or authenticating (to get a token issued). If the user is logging-in, in order to perform account management (see Section 2.4), she will receive a temporary session key which must be included in the signed queries to the partial IdPs for the rest of its account management-based REST calls in that session to ensure that these are also 2FA authorized.

Concretely we will consider one, or more, external Authenticator Providers (AuthProvs) that will facilitate the second factor authentication. We note that an Authenticator Provider could also be a partial IdP or a distributed vIdP. Whether it is distributed or not is depended on the 2FA implementation and deployment. Adding 2FA to an account involves the following steps:

1. The user will first sign in as normal to the vIdP, using its password.
2. The user will then contact, and register, at a AuthProv.
3. After doing so, it will perform an authentication with this AuthProv, which will result in a 2FA authentication token, $2Ftok$. This token *must* contain a unique identifier of the AuthProv, a unique user identifier, a time-stamp and a signature by the AuthProv, but can contain other information as well.
4. Finally the user will then use a *new* method in the account management interface, Add-2FA to add an AuthProv based on the token $2Ftok$, just received.
5. The partial IdPs will verify the signature on $2Ftok$ and that it has been constructed recently and that it stems from a trusted provider. If so, they will each update $\mathsf{acc}_i[uid]$ to contain the identity of the AuthProv and the user ID from $2Ftok$. That is, the initial $2Ftok$ is used to link a user in the vIdP to a user of a trusted AuthProv.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

After adding 2FA, the next time a user signs in or authenticates, it can start by performing a 2FA and include the time-constrained $2Ftok$ it gets back from the AuthProv in its request to the partial IdPs. To be able to manage 2FA on its account at the vIdP, it is clear that it must also be possible for the user to disable 2FA. This is done in a similar manner to authentication using a special method called Disable-2FA. Finally, the user should also be able to change its 2FA, which is done using another special method called Change-2FA. This method basically executed the steps of Disable-2FA and Add-2FA.

Concretely the methods that needs to be added to the account management interface can be described as follows:

Add 2FA $\mathcal{U}.\text{Add-2FA}(uid, 2Ftok, \sigma) \leftrightarrow$
$\mathcal{I}_i.\text{Add-2FA}(sk_i, uid, \text{acc}_i[uid]) : (\texttt{err}/ses), (\texttt{err}/\text{acc}_i[uid]')$.
Add 2FA is an interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid, a signed, valid 2FA token from a trusted AuthProv, $2Ftok$ and a signature on the message, σ, based on the user's private key (learned through $\mathcal{U}$.Login). The private input of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i and the user's current account info $\text{acc}_i[uid]$. Each partial identity provider verifies the existence of the user, her signature and the validity of the $2Ftok$. If the verification fails, an error message $\texttt{err}$ is output. Otherwise they return $\texttt{ok}$ and update the user's account info to contain the user ID from $2Ftok$ and the identifier of the AuthProv who produced the $2Ftok$. Denote the updated account info $\text{acc}_i[uid]'$ and have the partial IdPs return this to update their databases.

Disable 2FA $\mathcal{U}.\text{Disable-2FA}(uid, 2Ftok, \sigma) \leftrightarrow$
$\mathcal{I}_i.\text{Disable-2FA}(sk_i, uid, \text{acc}_i[uid]) : (\texttt{err}/\texttt{ok}), (\texttt{err}/\text{acc}_i[uid]')$.
Disable 2FA is an interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid, a signed, valid 2FA token from a trusted AuthProv, $2Ftok$ and a signature on the message, σ, based on the user's private key (learned through $\mathcal{U}$.Login). The private input of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i and the user's current account info $\text{acc}_i[uid]$. Each partial identity provider verifies existence of the user, her signature, that she has 2FA enabled and the token $2Ftok$ in accordance to the info stored $\text{acc}_i[uid]$. If verification fails, an error message $\texttt{err}$ is returned. Otherwise they return an output $\texttt{ok}$ and delete the 2FA info from the user's account info. Denote the updated account info $\text{acc}_i[uid]'$ and have the partial IdPs return this to update their databases.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

Change 2FA $\mathcal{U}.\mathsf{Change\text{-}2FA}(uid, 2Ftok, 2Ftok', \sigma) \leftrightarrow$
$\mathcal{I}_i.\mathsf{Change\text{-}2FA}(sk_i, uid, \mathsf{acc}_i[uid]) : (\mathtt{err}/\mathtt{ok}), (\mathtt{err}/\mathsf{acc}_i[uid]')$.

Change 2FA is an interactive protocol between the user $\mathcal{U}$ and all partial IdPs $\mathcal{I}_1, \ldots, \mathcal{I}_n$. The common input is the user name uid, two signed, valid 2FA tokens from trusted AuthProvs, $2Ftok$ and $2Ftok'$ and a signature on the message, σ, based on the user's private key (learned through $\mathcal{U}.\mathsf{Login}$). The private input of each partial identity provider $\mathcal{I}_i$ is his secret key share sk_i and the user's current account info $\mathsf{acc}_i[uid]$. The partial IdPs and the user executes the flow of "Disable 2FA" based on the information $2Ftok$ and if this goes well then it is followed by an execution of "Add 2FA" based on the new information $2Ftok'$.

Because of the extra $2Ftok$ in play when using 2FA, it is clear that the Login and Auth methods must be adapted to be able to take an extra argument from the user, that is, the $2Ftok$, value returned by the authenticator providing the 2FA. Furthermore it is needed that the Login function also return a temporary session key, ses, along with the user's private key sk. This session key is then added to all already existing methods in the account management interface, in order to allow all the partial IdPs to verify whether the user has 2FA authenticated for the given session.

In regards to the AuthProv, it must provide an interface for a user to register (based on a unique user ID), constructing a temporary token (containing the user ID, a time-stamp and a signature from the AuthProv) and an option for the user to manage its account.

We note that the AuthProv implementation is outside of the scope of OLYMPUS. However, the architecture described here could be used to work with the popular Google Authenticator. Concretely this could be achieved when *one* of the partial IdPs run as an AuthProv. This would involve the AuthProv sampling a unique and random user ID and then generating a QR code for a time-based OTP which can be displayed to the user using the Google Authenticator app. The user can scan the QR code to add it to its Google Authenticator app and send back the time-based token it displays to the AuthProv. The AuthProv verifies this and if so, it constructs a token $2Ftok$ with the time-stamp, the unique user ID it picked and its signature, which it returns to the user-client. The client can then include this when calling Add-2FA towards all the servers. In fact any standard method for 2FA could be used for this by having a partial IdP act as an adapter between the 2FA mechanism and the rest of the OLYMPUS system. All that is required for seamless integration is that the client is able to handle the 2FA chosen.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

We also note that our approach to 2FA is only meant to protect against an *external* adversary who might have gotten a hold of the user's password through some side-channel. What we mean by an external adversary in this context is one that cannot corrupt one of the partial IdPs. While the solution can allow for partial IdPs to be corrupted, it will require the user to perform the 2FA protocol with each individual partial IdP, offering poor usability. In the Google Authenticator example above we note that only a single partial IdP acts as AuthProv and thus if this one gets corrupted then the 2FA adds no *further* security. Thus only a single corrupt partial IdP (if it is the one performing the 2FA) is enough to impersonate a user *if* it has learned the user's password through a side-channel since it can pretend to be this user towards the other partial IdPs. If we wish to protect against a partial IdP who has learned the user's password then we must have *all* partial IdPs *each* perform the 2FA with the user (and not share the result with the other partial IdPs). In that case, if at most $n-1$ partial IdPs are corrupted, they will not be able to impersonate the user towards the last honest partial IdP as they will not be able to guess the honest partial IdP's result for 2FA.

Security Requirements Our approach to 2FA allows the OLYMPUS system to integrate with another, independent authentication mechanism provided by an AuthProv (which may also be distributed). For this reason the added security depends on the specific realization of the 2FA.

- If at most $n-1$ IdPs are compromised:
  - **2FA-SReq-1:** None of the messages sent between the users, AuthProv and partial IdPs must contain information that allows to offline attack the user's password attempt.
  - **2FA-SReq-2:** A compromised AuthProv must not make it easier for an adversary to carry out an offline attack on a user's password.
  - **2FA-SReq-3:** If the AuthProv is not a partial IdP then a compromise of it must not allow leakage of any of the user's private information stored at the partial IdPs.
  - **2FA-SReq-4:** A user can only add a second factor to an account where she knows the account password.
  - **2FA-SReq-5:** A user can only provide a 2FA for an account where she knows the password.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

– **2FA-SReq-6:** An attacker can only perform an online attack on a user's 2FA account if it knows the user's password.

2.6 Refresh and backup

To protect against the scenario where an adversary iteratively corrupts all n partial IdPs through a prolonged period of time and hence getting *proactive security* we support the refreshing of cryptographic material of each partial IdP at regular intervals. This setting however requires offline storage of some master secret material for each of the partial IdPs which is not used during regular operation but is needed during refreshing. The refreshing only involves the partial IdPs and hence the users will be oblivious of such refreshing as the common vIdP public key will remain the same. A refresh procedure invalidates the secret key used during operation and thus ensures that in order to break the OLYMPUS system an adversary must compromise *all* partial IdPs in the time interval between two refresh procedures; which we call an *epoch*.

For each epoch, each partial IdP has some secret epoch key associated, denoted by sk_i for partial IdP $\mathcal{I}_i$, along with some master secret key material, denoted ms_i for partial IdP $\mathcal{I}_i$. The secret epoch key sk_i is used for every registration, authentication, and login query received by users during that epoch. To end an epoch and start a new one a cryptographic method is carried out on ms_i which produces a new epoch key sk_i', which replaces the old epoch key sk_i. The master secret ms_i is *not* used during any such queries and *must* be stored on a medium not connected to the internet. It is only used during setup and refresh, and in none of the cases will it ever be present in a database or on a physical disc. However, it is *very* important to store an offline backup of this master key in case of malicious corruption of the other partial IdPs. Such a backup will allow one to setup the system again using refreshed epoch keys without the need for any interaction with the users.

Key refreshing is carried out by a special REST call to each of the partial IdPs at the same time. The call contains a special refresh key key from the user which each partial IdP will verify against a reference value in their database. If the check goes through then the partial IdPs will start an interactive protocol to update their current epoch key sk_i based on all the partial IdPs' respective master secret shares ms_i. The result is a new epoch key, sk_i', that will replace the old one and used for all future user queries. Alternatively a local service could be

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

implemented on each partial IdP that initiates the key refreshing step locally at specific, predefined times.

We note that each partial IdP's master secret ms_i is *never* stored locally, persistently by each IdP, but is instead secret shared between all the partial IdPs. Hence a subprotocol is carried out between the partial IdPs to allow each of them to retrieve their master key and keep it in RAM *only* during the refresh procedure. To ensure that the master key has been correctly restored, a hash digest of it is saved locally in each server's database. We note that this does not leak any information on the key itself since it is high entropy. If the digest check fails, this means that one of the servers have been acting maliciously, or that the server's database has been compromised (and the digest value changed). In this case a physical backup of the master key used during setup of the system must be retrieved and the setup procedure must be carried out again (once the system administrator have made sure that the threat that caused the compromise has been removed). Even in this case, the users do not need to do anything and the system can function as before the refresh procedure.

Concretely refreshing is done using the following REST interface[2]:

$\mathcal{U}$.Refresh$(key, key') \leftrightarrow$
$\mathcal{I}_i$.Refresh$(ms_i, key) : (\texttt{err}/\texttt{ok}), (\texttt{err}/sk_i')$.

Refreshing is an interactive protocol between all the partial IdPs, $\mathcal{I}_1, \ldots, \mathcal{I}_n$, but is triggered by a REST call from a user $\mathcal{U}$. The user's private input is a special refresh key, key, and an optional new refresh key key' to replace key after successful completion. The private input of each of the partial IdPs $\mathcal{I}_i$ is his master key share ms_i and the locally stored special refresh key key. At the end of the protocol the user receives either an error message, `err` or a success message, `ok` message indicating whether the procedure was successful. Each partial IdP $\mathcal{I}_i$ either receives an error message, `err` or a new secret epoch key sk_i', which it will use for all queries in the future. If desired, the error message can give an indication why the protocol failed. Possible reasons could be an incorrect refresh key key.

We note that the server will be in an "inconsistent" state during refreshing and hence will not be able to correctly respond to user queries until the refresh has been carried out. If a user were to try to authenticate

[2]We note here that the user $\mathcal{U}$ is a special user, which is an administrator. However we use this notation since the refresh is issued as a simple REST call like normal user queries.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

while the refresh is happening she would learn some inconsistent information which, when verified by the partial IdPs, would result in them thinking that an incorrect password was produced.

Since such a refresh will not happen often and only last very few seconds, we currently accept this problem and will instead direct the user to try again in case authentication fails on a supposedly correct password.

Because of this brief unavailability of the system during refresh, we require a special refresh key, key, to start the refresh procedure. Still, this key is not more critical than it can simply be stored in each partial IdP's regular database, as it purely is used to protect against a flood of refresh calls from a malicious client, which would result in a denial of service attack. Still, this is all an adversary with knowledge of this key can achieve; knowledge of key will not allow it to extract any secret information or destroy any data.

We have formalized how such a refresh procedure is carried out for the Distributed Password Verification (DPV) protocol and the Distributed Token Generation protocol (when the token is a JWT token based on a distributed RSA signature) in the PESTO paper [2] and Deliverable 4.1. Unfortunately, due to technical reasons, it is not possible to implement such a proactive key refreshing mechanism using our existing distributed credential (distributed P-ABC) scheme. In such a setting all the users' credentials must be reissued *and* the verification key, used by the Service Providers to verify the signature on a credential, must also be updated. For this reason we have limited the proactive security to the DPV mechanism and the DTG based on RSA signatures.

Security Requirements Our approach to refreshing is constructed to allow OLYMPUS to keep running securely even if some of the partial IdPs have been compromised (for the case where distributed RSA is used for token generation).

- If at most $n - 1$ IdPs are compromised at any single point in time (but their master keys remain secure) and distributed RSA is used for token generation:
  - **Refresh-SReq-1:** None of the data of a partial IdP will provide any advantage in bruteforcing a user's password or impersonating the vIdP in a future epoch. This is so even if the remaining partial IdP is compromised in a future epoch.
  - **Refresh-SReq-2:** An adversary must not easily be able to

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

prevent usage of the system by honest users.

2.7 Compliance with EU regulation

GDPR compliance As the vIdP in OLYMPUS will be dealing with personal data the GDPR compliance must be addressed. Deliverable D3.2 discusses how the OLYMPUS project deals with these issues from a general perspective. Nevertheless, each particular deployment might involve different implications in the light of the GDPR, therefore a second context-based approach must be addressed.

With regard to the core elements of OLYMPUS we must point out the following reflections.

1. OLYMPUS will assure the respect of data minimization and proportionality in the data processing. This last aspect is currently being investigated under possibilities such as blind/ committed storage of attributes as explained in section 2.9 of this document.
2. ID proofing is out of the scope and the accuracy of the data will be guaranteed in a moment prior to storage through an external valid IdP. Nevertheless, the requirement of accuracy also involves the right to update or modify the data in a later moment. This is addressed in OLYMPUS by the functions of “delete attribute” and “add attribute”, which at the same time will guarantee the accuracy of the data modified requiring a new ID Proofing in order to add the new attribute.
3. OLYMPUS will guarantee transparency in the processing which directly affects the right of access. On the one hand this right refers to the LEA and on the other to the user, so data subject has the right to access their own data, as well as to know how their data are being processed and for what purpose/ purposes. This last aspect can be conveniently covered by a privacy policy, while the right to access is partially covered by OLYMPUS function of “retrieve information” which allows the user to receive all stored attributes in a suitable format for export/ import with other services. It should be noted however, that this function might be more appropriate for the exercise of the user’s right to data portability.
4. OLYMPUS will guarantee the right to be forgotten through the right to erasure. This right is partially accommodated by the functions “delete attribute” or “delete account”. This accommodation is only partial because data stored outside of the vIdP database

This project has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 786725

e.g in logs or backup cannot be manipulated by the OLYMPUS core and must therefore be deleted using a different mechanism, specific for the concrete deployment. The same reflections can be done with regard to double storage (e.g. in the use cases), as the right to be forgotten requires permanent deletion of all of them.

5. OLYMPUS core elements assure data confidentiallity by the implementation of complex cryptographic protocols that protect from identity-related crime focused on the IdP, combined with 2F authentication, secure connections and the possible implementation of blind/ committed storage of attributes to protect before different kind of attacks.

eIDAS Regulation With regard to the core elements in OLYMPUS it is also necessary to address whether they fulfill the technical requirements to be considered pursuing the eIDAS Regulation as an identification system of level substantial or high, and therefore being of compulsory mutual recognition.

The Article 7 of the eIDAS Regulation establishes a set of conditions for the mutual recognition of the electronic identification means in the European Union. Among these conditions, pursuing sections b) and c) “the assurance level must be equal or higher than the level required by the relevant public sector body to access the service online” and “the relevant public sector body uses the assurance level substantial or high in relation to accessing the service online”. In consequence, in the case of low-level systems, such recognition is optional and therefore it will depend on the agreement to which the Member States may come with other Member States.

Security levels are described in Article 8.2, as a number of high-level and somewhat abstract criteria that support a particular degree of confidence in the electronic identification means issued to the person, while reducing or avoiding the risk of misuse or undue alteration of identity. These levels differ according to the risk of use of the electronic identification in a specific service, that is, depending on the probability of occurrence of a threat, with the qualitatively or quantitatively determinable harmful impact, and that usually correspond to what in the standards is called “levels or authentication assurance”.

To determine whether an electronic mean fulfills these LoAs, the Annex of the Commission Implementing Regulation (EU) 2015/1502 of 8th September 2015 on setting out minimum technical specifications and procedures for assurance levels for electronic identification means,

This project has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 786725

sets the elements of technical specifications and procedures to determine how the requirements and criteria of Article 8 of the eIDAS Regulation shall be applied.

The Annex distinguishes between enrolment and authentication phases as well as management tasks. The two first phases are directly affected by OLYMPUS core elements, while the management tasks are context-based and rely more on the legal entity finally deploying the OLYMPUS technology.

Enrolment *Registration*. The user must be aware of terms and conditions, as well as of recommended security precautions. This might be relevant in the case of pABCs in OLYMPUS, as the user must have knowledge of recommended precautions regarding the material stored in his mobile device. Finally, the electronic identification mean shall collect the relevant data required for the ID proofing, which in the case of OLYMPUS consists in the credentials (username and password), that once issued allow to carry the ID proofing consecutively or in a later moment in time.

ID Proofing. This phase is out of the scope in OLYMPUS as it is envisaged that the ID Proofing will be carried out by an external IdP which will assert the data (and that must therefore fulfill the requirements regarding to this phase in the Commission Implementing Regulation 2015/1502) before being encrypted and stored by the OLYMPUS IdP. Nevertheless, this aspect must be readdressed in each particular deployment, as it relies on aspects such as whether the IdPs exporting and importing the attributes are or not the same legal entity/organization.

Management of the electronic identification mean. The level substantial requires at least two authentication factors from different categories. This has been addressed in section 2.5. Furthermore, OLYMPUS core elements will reach the level of high assurance considering the improvements introduced in the prevention from attackers of high potential as well as from attacks aiming to impersonate the user, or in other words, it protects from the use by others.

Issuance, delivery and activation. The electronic identification means are delivered via a mechanism by which it can be assumed to be delivered to the person to whom it belongs. In the case of OLYMPUS, the person who has proved to have knowledge of the previously issued credentials. To reach the high level of assurance, an activation process would be necessary in order to verify the identity (e.g. a text

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

message or an email).

Suspension, revocation and reactivation. It is possible to suspend or revoke the electronic identification mean in a timely and effective manner and there are measures to prevent from unauthorized access. In OLYMPUS the person has to prove to have knowledge of the credentials to carry out these operations, hence, the same level of assurance as with authentication is offered.

Renewal and replacement. At least need to meet the same requirements with initial ID proofing. OLYMPUS fulfills this requirement as in case of renewal or replacement of the account, where it would be necessary to repeat the process of registration. Likewise, in the case of user's attributes, after the attribute deletion, in order to add new attributes a new ID proofing must be carried out.

Authentication *Authentication mechanism*. OLYMPUS will fulfill the requirements for levels substantial and high as it involves 2F in authentication and it protects before attackers with moderate or even high potential as it is very unlikely that attackers with high potential might be able to subvert the authentication mechanism.

A short reflection on the possibilities of OLYMPUS to foster a new drafting of the eIDAS Regulation Although OLYMPUS has introduced in this deliverable as part of its core elements a multi-factor authentication, original design had not envisaged the implementation of 2F because the password-based authentication mechanism combined with the cryptographic protocols deployed would have been enough to prevent from a wide range of attacks. Hence, although this has finally been modified, it is clear that there exists a regulatory barrier as it excludes all electronic identification means which are not based on multi-factor authentication. This would also contravene the principle of technological neutrality as organizations and individuals could not use different electronic identification means from multi-factor authentication methods for cross-border operations, contravening what stated in Recital 16 of the eIDAS Regulation, "the requirements established should be technology-neutral" and "it should be possible to achieve the necessary security requirements through different technologies". Likewise, section 3 of the Article 12 of the eIDAS Regulation states that "it aims to be technology neutral and does not discriminate between any specific national technical solutions for electronic identification within a Member State where possible". In consequence, it is clear that it is necessary to review this requirement and adapt it to nowadays tech-

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

nological possibilities to allow different designs which offer the same or even a higher protection that multi-factor authentication.

2.8 Interfaces of Components and message formats

We summarize the required interfaces for the core functionality of the main components of the OLYMPUS system in Figure 3. As mentioned earlier, the "Login / OPRF" functionality is used to obtain the user's private key, which is used to produce the signatures used in the following messages.

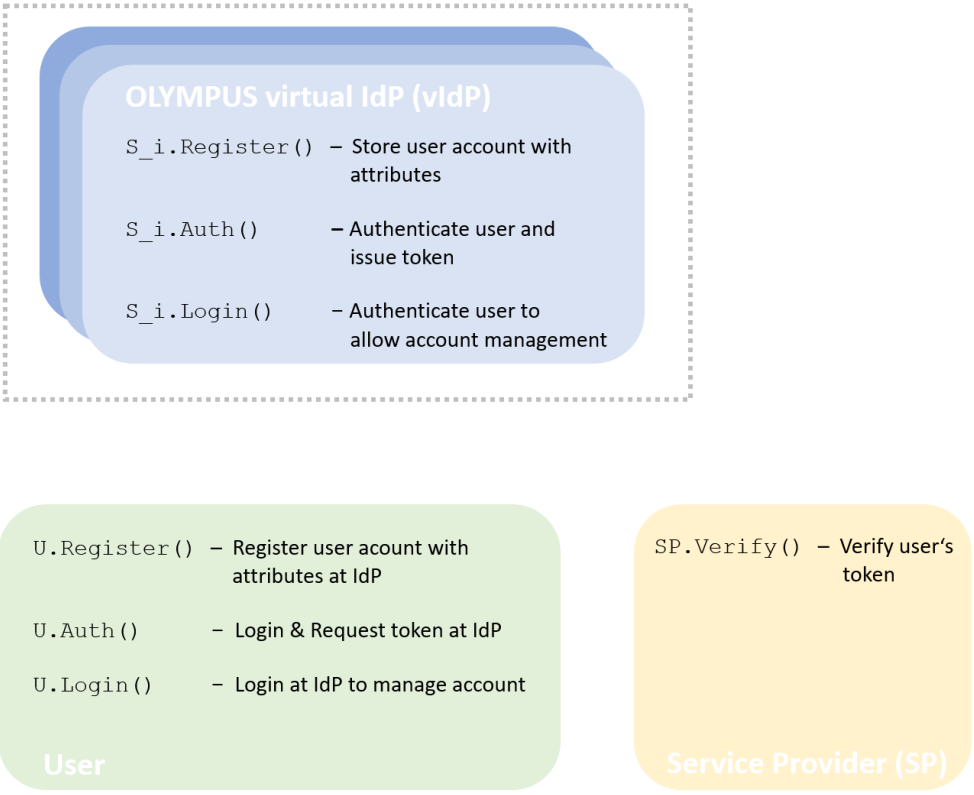


Figure 3: Interfaces of core components of the OLYMPUS system.

Login / OPRF The client makes an HTTP POST call to https://address:port/idp/requestOPRF, using the following JSON body:

- ssid: A unique session identifier.
- username: The uid.
- element: The Base64 encoding of an eliptic curve point.

An example of the JSON message:

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

```
{  "ssid": "1589891566840",
    "username": "user_1",
    "element": "BBHJtu0OoiVUlWI5wPLouIAAAAAAAAAAAA"
}
```

The IdP responds with a similar JSON message, containing the ssid and a Base64 encoding of another eliptic curve point:

```
{  "ssid": "1589891566840",
    "element": "D7TY0gVRgi+gvZ2epubxcJ18rlh1LR1/C7"
}
```

Registration The client finalizes the registration by making an HTTP POST to https://address:port/idp/finishRegistration, using the following JSON body:

- username: The uid.
- publicKey: An encoding of the users public key.
- signature: A Base64 encoding of the signature on the uid and nonce.
- ssid: The ssid of the session (will be used to generate the nonce).
- idProof: The optional set of user attributes.

An example of the JSON message:

```
{  "username": "user_1",
    "publicKey": {
      "algorithm": "EC",
      "format": "X.509",
      "encoded": "MFkwEwYHKoZIzj0CYIKoZIzj0...g==",
      "bytes":"WC41MDlFQ01Ga3dFdlIS29baSXpq...AAA"
    },
    "signature": "MEQCICWkmVDZ2JrLfetxl1idy4Ug==",
    "ssid":1589891566840,
    "idProof":{"optional"}}
```

Assuming the registration went well, the IdPs will all respond with a Base64 encoded signature on the users public key, e.g.:

```
mPHBPjpGeFQ+uK8cFDLhSw1mllO+Ngifn5vG/...cfXUgg==
```

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

Authentication The client authenticates by making an HTTP POST to https://address:port/idp/authenticate, using the following JSON body:

- username: The uid.
- ssid: The ssid of the session (will be used to generate the nonce).
- signature: A Base64 encoding of the signature on the uid and nonce.
- policy: The policy the issued token should satisfy.

An example of the JSON message:

```
{"username": "user_1",
    "ssid": 1589985153251,
    "signature": "MEUCICzDhaIUieDbR09uuQJ3AH...c=",
    "policy": {...}}
```

Issue pABC The client makes an HTTP POST to https://address:port/idp/getCredentialShare, using the following JSON body:

- username: The uid.
- ssid: The ssid of the session (will be used to generate the nonce).
- signature: A Base64 encoding of the signature on the uid and nonce.

An example of the JSON message:

```
{"username": "user_1",
    "ssid": 1589995183221,
    "signature": "MEUCI2aDhaIUUeBbR09uuQJ3AH...c=",
```

Retrieve information The client makes an HTTP POST to https://address:port/idp/getAllAttributes, using the following JSON body:

- username: The uid.
- ssid: The ssid of the session (will be used to generate the nonce).
- signature: A Base64 encoding of the signature on the uid and nonce.

An example of the JSON message:

```
{"username": "user_1",
    "ssid": 1589985153252,
```

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

```
3     "signature": "MEQCIFa0Hx5hwyzFtWJG06CWK...g=="}
```

Add attribute / identity proving: The client makes an HTTP POST to https://address:port/idp/proveId, using the following JSON body:

- username: The uid.
- ssid: The ssid of the session (will be used to generate the nonce).
- signature: A Base64 encoding of the signature on the uid and nonce.
- idProof: An identity proof containing the attibutes to add to the IdP.

An example of the JSON message:

```
1 { "username": "user_1",
2     "ssid": 1590400951188,
3     "signature": "MEQCIQD3Jftr2R/UniqNvf2nTmQOH...A
          ==",
4     "idProof": {...}
5 }
```

Delete attributes: The client makes an HTTP POST to https://address:port/idp/deleteAttributes, using the following JSON body:

- username: The uid.
- ssid: The ssid of the session (will be used to generate the nonce).
- signature: A Base64 encoding of the signature on the uid and nonce.
- attributes: A list of attribute names to delete.

An example of the JSON message:

```
1 { "username": "user_1",
2     "ssid": 159040260045,
3     "signature": "MEUCIQCsJW5OMf18X/igJqXHLB...=",
4     "attributes": ["Name", "Age"]
5 }
```

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

Delete account: The client makes an HTTP POST to https://address:port/idp/deleteAccount, using the following JSON body:

- username: The uid.
- ssid: The ssid of the session (will be used to generate the nonce).
- signature: A Base64 encoding of the signature on the uid and nonce.

An example of the JSON message:

```
{ "username": "user_1",
    "ssid": 1590404876068,
    "signature": "MEUCIEMofv9NfrZQHMazNIP7f+w...="
}
```

Change password: The client runs the OPRF protocol twice, using the old and new password, in order to obtain the old and new private key. The client then makes an HTTP POST to https://address:port/idp/changePassword, using the following JSON body:

- username: The uid.
- publicKey: An encoding of the users new public key.
- ssid: The ssid of the session (will be used to generate the nonce).
- newSignature: A Base64 encoding of the signature on the uid, nonce and publicKey, using the new private key.
- oldSignature: A Base64 encoding of the signature on newSignature, using the old private key.
- signature: A Base64 encoding of the signature on the uid and nonce.

An example of the JSON message:

```
{  "username": "user_1",
    "publicKey": {
      "algorithm": "EC",
      "format": "X.509",
      "encoded": "MFkwEwYHKoZIzj0CYIKoZIzj0...g==",
      "bytes":"WC41MDlFQ01Ga3dFdlIS29baSXpq...AAA"
    },
    "oldSignature": "MEYCIQCULYdhxjzPJLGHf...k7R",
    "newSignature": "MEUCIFJUnWk/eAcerucpv...qA=",
```

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

```
10      "ssid":1590405223439}
```

2.9 Open Design Question

The core OLYMPUS system as described in this section can be augmented in several ways. We now provide an outlook on properties that would be desireable to integrate. Some of them are still investigated within the runtime of the project.

Blind/Committed Storage of Attributes In the currently described architecture the user's attributes are stored at the vIdP in the clear. This means that each physical IdP holds a copy of the user's attributes, which is not ideal from a privacy point of view. We plan to investigate how blind signatures and cryptographic commitments can be used to let the vIdP issue token on hidden attributes.

Attribute Aggregation It would be interesting to investigate whether an efficient solution exists that lets physical IdPs hold different attributes. The joint token generation would then have to perform aggregation of attributes collected at all separate physical IdPs. Such a solution would also enhance the privacy of the user since not all servers need to know all attributes. Using techniques from anonymization, we could even hope to protect the real identity of the user despite the fact that knowledge of all attributes would reveal it.

3. Deployment

As the OLYMPUS framework introduces components for all 3 roles in a Federated Identity system, i.e. user, IdP and service provider, there are a number of possible deployment models that can be used:

3.1 User side deployment

In a traditional IdP deployment, the IdP consists of a single server which a client interacts with, using a fairly simple interface. The typical IdP uses a simple *request-response* protocol, where the client sends a single message to the server and gets an access token (or rejection) in return.

Due to the nature of the virtual IdP, the OLYMPUS interface becomes more advanced and may require multiple rounds of communication be-

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

tween the client and IdP server. The more advanced protocols therefore require some logic to be executed on the client side. Traditionally such client logic could be implemented in JavaScript and be hosted by the IdP (or potentially by the service provider), however such hosting would introduce a new single-point-of-failure. Instead we describe some other possible deployments:

Code based integration For smartphone and other non-browser based user applications, the client logic can be implemented directly in the application as a third-party (with regard to the application) code library. The library will offer a simple API for functionality, e.g. *registration* and *authentication*, relevant for user clients as mentioned in Section 2.

The methods exposed by the API will take username, password and relevant parameters such as an access policy as input and produce the corresponding output. The user application may remain oblivious to the cryptographic protocols executed by the library and may thus focus on the application logic. However, if the application desires to use offline authentication (based on p-ABCs), a secure storage for the credentials will be needed. In that case, even if the laptop/smartphone does not have Internet connection, but has another way to communicate with the verifier (e.g. Bluetooth), the application can still use the API to generate an access token (other functionality, like account management or retrieving new credentials after the stored one expires, will of course not be possible).

Front-end / **gateway** If a rich client application is not possible, the OLYMPUS code library may also be wrapped in a web-server container. This allows the OLYMPUS functionality to be deployed in a manner not requiring special logic to be executed directly in the client, but rather on the local gateway/front-end server. Essentially this deployment model allows the gateway to expose a standard IdP API to the client, requiring no changes to the clients, but handling advanced functionality by the gateway. The gateway may be deployed directly on a user laptop or similar, only accepting local communication, or on a server owned by the user's organization.

In this gateway/front-end scenario, the offline use case is not as clear. If the gateway is local to the user device, it could in principle manage the credential storage and offer the same functionality as in the code integration, however in practice it is unlikely that this would be benificial compared to a code based integration. Contrarily, if the gateway is deployed in a server owned by the user's organization, the whole concept of an offline use case becomes questionable as the user device

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

would need to be able to contact this server.

Note that if the gateway is accessible by multiple users, it becomes a single-point-of-failure / single-point-of-attack similar to standard IdP solution. This is unavoidable since the gateway will process both the users passwords and the resulting access tokens. The deploying organization does however maintain control of the gateway, keeping it internal within the organization, while still utilizing the external vIdP.

Browser plugin As a hybrid solution utilizing both of the described deployments, the client logic may also be implemented as a browser plugin. This will allow for the OLYMPUS client code to be deployed directly in the browser, allowing existing web applications to utilize the offered functionality of OLYMPUS without needing to integrate the OLYMPUS client directly.

In this setting the browser plugin would need to monitor all HTTP requests of the browser and intercept and take over any requests to a OLYMPUS vIdP. Since the client does not need storage and the logic is fairly simple, this approach does seem feasible in a secure manner. The distributed authentication does however rely on bi-linear pairings as a cryptographic primitive. While JavaScript implementations of these primitives do exist, they are not widely used, hence additional implementation work would be needed and the offered performance may not be able to meet the requirements of a full OLYMPUS deployment. Also, this approach rules out the offline authentication use case, as secure storage is required and is not currently offered in relation to browser plugins.

3.2 vIdP deployment

Deploying an OLYMPUS vIdP is comparable to deploying a traditional IdP, except 2 or more server instances must be deployed. The IdP application itself must be configured and built as described in deliverable D4.2, however once the application has been built, it can be deployed like any other web application.

The framework comes with a default webserver, which allows the IdP server application to be run as a stand-alone application. Alternatively the application can be deployed in a standard web container such as Tomcat.

By default the OLYMPUS client and vIdP communicate using REST, hence communication using REST must be allowed in the relevant

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

firewalls. If the deployment can not rely on REST, a custom communication layer must be implemented between the client and IdPs.

As part of the IdP configuration and building process, certain steps must be taken to integrate the partial IdPs with existing infrastructure. This can be boiled down to 3 components that must be implemented:

Database integration In order to persist data, the IdP connects with some database. In order to do so, the application must implement a database interface and provide the implementation to the server application. The interface is oblivious to how data is actually stored and just requires a few basic methods for user management, such as storing and reading of user attributes. The framework comes with a reference implementation of an *in-memory* database, which can be used for prototyping, but will reset when the server application is terminated.

Hardware security If the deployment requires the use of a *hardware security module* (HSM), the application must implement a *CryptoModule* interface and provide that to the server application. The framework comes with a software based implementation that may be used in most cases. If there is a need, that implementation can be overridden and a custom hardware based implementation can be used instead.

Identity proving As described in Section 2.2 the validity of the user attributes must be ensured. As user attributes may originate from many resources, i.e. other IdPs, eIDs or ad-hoc identity solutions, the core OLYMPUS framework does not validate the attributes directly. Instead a flexible approach is taken, where a vIdP deployment must implement one or more Identity Provers, responsible for validating the attributes entering the vIdP. This allows each deployment to support the data formats specific for the attribute resources used in that deployment.

For instance, one deployment may use passport data and will therefore implement a code components for checking the signature and format of data representing a digital passport. Another deployment may instead accept signed XML documents containing financial information and would therefore implement a code component handling XML processing and validation.

This approach allows a given deployment to utilize any attribute resource required, although it comes at the cost of some implementation work specific for the deployment.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

3.3 Service provider deployment

One of the goals of the OLYMPUS project, is to achieve a degree of backwards compatibility in the sense that existing service providers should be minimaly affected. As such, the access tokens produced by the vIdP are based on common signature schemes RSA or ECDSA and should be formed according to some commonly used solution such as JWT or SAML. This will allow the vIdP to easily fit into OAuth and similar flows, such as OpenID Connect, without any OLYMPUS specific requirements on the SP. This allows the Service Provice to use a COTS product for verifying the access token.

The service provider will of course need to add public key material, most likely in the form of an X509 certificate, to the truststore. Since the certificate does not contain sensitive information and may be considered public knowledge, each partial IdP, will have a copy of the public key material. Furthermore, the key material will, in most deployments, be a regular RSA key, allowing common PKI infrastructure to be used.

In the case of DP-ABC, the access token sent to the service provider is however not based on a common signature scheme and COTS products can therefore not be used for verifying a DP-ABC-based token. In order to verify a DP-ABC based token, an OLYMPUS verifier library should be used. This library can either be implemented directly in the Service Provider application or a REST based verifier service can be hosted (internally) as part of the Service Provider deployment. As in the signature based scheme, the Service Provider will need public key material from the vIdP (and some public parameters). Again this key material can be obtained from any partial IdP, although the DP-ABC specific key material may not fit in a X509 format due to the nature of the key material.

An open question with regard to the SP deployment and integration, is how the access requirements, i.e. what information should the access token contain, is communicated to the user client. The OLYMPUS user client API requires this information in an internal OLYMPUS format, hence the deployment specific user client implementation is responsible for translating the used format into the format used internally in OLYMPUS. The internal format may also be used externally as a proprietary format. If another more standard policy format is required, the client application is responsible for translating this into the OLYMPUS format.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

3.4 OpenID connect support

The sketched deployments can be used to support standard identity management schemes such as an OIDC authorisation flow by adapting the transmitted artifacts:

The standard OIDC flow starts with the user visiting a service provider web page and choosing an IdP. This allows the service provider to produce a HTTP redirect URL, containing the ID of the service provider and a list of claims to reveal. The user follows the redirect to the IdP, which authenticates the user and produces a JWT access token along with the HTTP redirect back to the service provider. By following the HTTP redirect, the user transmits the JWT to the service provider and thereby authenticates with the service provider.

The OIDC flow could be supported by the OLYMPUS framework by using the browser plugin based user deployment model. By having the browser plugin listen for HTTP redirects, the user could use the OLYMPUS based protocols to authenticate the user towards the vIdP rather than the traditional IdP. Once the protocol is completed, the browser plugin would redirect (with the redirect URL containing the resulting JWT token) back to the service provider. At this point the user agent/browser would be completely oblivious to the fact that the vIdP had been used rather than a traditional IdP.

Note that in order to support the OIDC flow, it is necessary for the IdP to known the identity of the service provider. This is to ensure that the JWT token is bound to a specific SP and is not used by a malicious SP to impersonate the user towards a different SP. This may violate an unlinkability requirement of OLYMPUS, but is necessary to comply with the OIDC flow. A possible solution for this may be blind signatures (left for future work).

Also note that the sketched OIDC flow uses the *implicit flow* grant type. Other grant types work slightly different, e.g. in the *authentication flow* grant type, the user passes a one-time token to the service provider, which the service provider exchanges to a proper token at the IdP. These other grant types may be supported, however it may have implications on the security model, e.g. if *authentication flow* is used, it will be impossible to prevent linkability between the SP and IdP.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

4. Instantiating the components

From a cryptographic point of view, the authentication process allowing a user to login with a previously registered password to obtain a signed access token consists of two main parts:

- Distributed password verification (DPV)
- Distributed generation of the access token

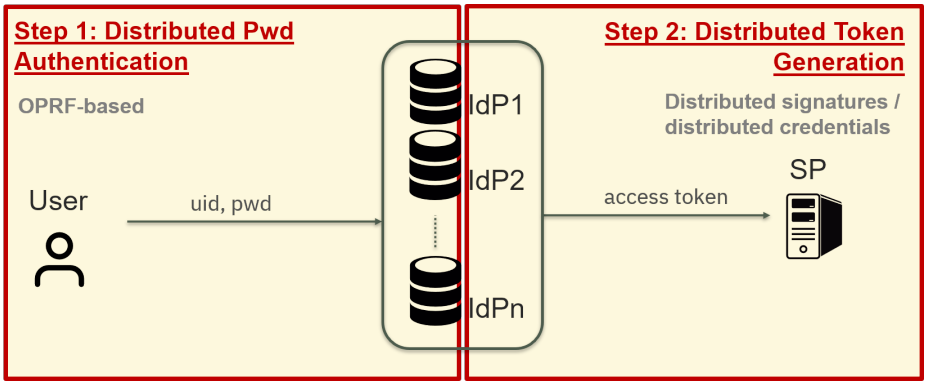


Figure 4: High-level overview of our protocols.

Instantiations of these parts are described in deliverable D4.1. In the following we sketch the high-level ideas of our protocols and security properties.

Please note that ideas sketched here are mainly for illustrative purposes and omit several steps that are needed for achieving the desired security goals. For the detailed description we refer to D4.1

4.1 Distributed Password Verification

Our n IdPs must be able to obtain information derived from a user password pwd that allows them to later verify whether for a password attempt pwd' it holds that $pwd = pwd'$.

The core of our protocol is a *partially-oblivious and distributed pseudorandom function* (dpOPRF) that allows n parties to blindly and cooperatively compute a value $h = \mathrm{PRF}(k, (uid, pwd))$. The key k is distributed in n shares $k_1, \ldots, k_n$ among all IdPs and the obliviousness of the dpOPRF allows each IdP to blindly compute their evaluation share for a blinded input derived from pwd and a public input uid.

The user can combine and unblind these shares into the full output $h = \mathrm{PRF}(k, (uid, pwd))$ from which she deterministically derives a key pair

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

(usk, upk) of a standard signature scheme. The user finally registers (uid, upk) with all IdPs.

When the user wants to authenticate for uid with password attempt pwd', she simply runs the same dpOPRF protocol obtaining $h' = \mathsf{PRF}(k, (uid, pwd'))$. The user then re-computes usk from h', signs a fresh nonce under usk and sends the signature to all servers which finally verify the received signature against the upk that is stored for uid.

Session ID For security reasons it is necessary that all cryptographic operations between a user and the partial IdPs are carried out over a unique session ID. Concretely this session ID is computed non-interactively as the hash digest of the requesting user's ID and the current time-stamp (sampled by the user). To ensure uniqueness, the partial IdPs save the last time-stamp used and will only allow usage of a newer time-stamp, but only within 10 seconds of their wall-call time.

Security Discussion We discuss how the different security requirements are fulfilled by the distributed password verification approach.

We first consider the registration requirements as defined in Section 2.2 and all are under the assumption that at most $n - 1$ partial IdPs are compromised.

Reg-SReq-1 : This requirement states that the user's messages should not allow for an offline attack on its password. This is fulfilled given that the user's password is only used through the dpOPRF, which by cryptographic construction ensures that a polynomially bounded server (in the security parameter) cannot learn the input assuming a Diffie-Hellman style assumption.

Reg-SReq-2 : This requirement states that the data saved on a user by the IdPs also should not allow for an offline attack on it's password. This is also fulfilled since the only data dependent on the user's password, which the servers store, is based on the output of the dpOPRF, but by cryptographic construction, the output of the dpOPRF is indistinguishable from random (when less than all partial IdPs private keys are known) and thus does not leak anything about the user's input/password either.

Reg-SReq-3 : This requirement states that an adversary can be removed from the partial IdPs without requiring all users to perform registration again. As discussed in Deliverable 4.1 this is handled in the Distributed Password Verification approach by a

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

refresh mechanism that allows all partial IdPs to refresh their private keys and hence an adversary having keys of all but one of the partial IdPs before a refresh will not be able to use these afterwards. Given that the refresh is only of the partial IdPs' private keys (which has nothing to do with the user) there is no need for the user to do anything in relation to a refresh.

We next consider the requirements for authentication as defined in Section 2.3, again under the assumption that at most $n - 1$ partial IdPs are corrupted.

Auth-SReq-1 : This requirement states that the user's messages should not allow for an offline attack on its password. Given that both registration and authentication only requires the user to use its password as input to the dpOPRF, this requirement is fulfilled by the same argument as for **Ref-SReq-1**.

Auth-SReq-2 : This requirement states that a user must know the password used through registration to be able to authenticate. This is fulfilled since the output of the dpOPRF is used to construct a signing key that the user must use on a challenge to prove it knows its password, where its corresponding verification key has been stored at the partial IdPs. The output of the dpOPRF (and hence the signing key) is determined from the user's password in a non-invertible way, thus the password is required to restore the signing key, which is required to authenticate. So unless the user is sharing/storing the signing key, which is *not* intended behaviour, or an adversary extracts it from device memory during authentication, then it is not possible to impersonate the user without knowing its password.

Auth-SReq-3 : This requirement states that the IdPs must be able to limit online attacks. Given that the dpOPRF "leaks" the user's ID to all the partial identifiers they know the amount of attempts. Furthermore, given that the user must return a signed challenge the partial IdPs will also learn whether an authentication attempt was successfully completed and throttle authentication attempts accordingly based on this information (and hence not allow an adversary to repeatedly try passwords for a specific user).

Auth-SReq-4 *optional*: This requirement states the adversary must not learn the access policy of an authentication request. Currently this is *not* fulfilled for the token signatures scheme, since the partial IdPs sign a token in plain which contains the user's access policy. This is needed in order for them to verify that a

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

given user indeed fulfils a certain policy. When using dp-ABCs, the access policy never leaves the client, hence the adversary can never learn it.

Auth-SReq-5 *optional*: This requirement states that the adversary does not learn anything about the access token output to the user. Similar to **Auth-SREQ-4**, this is currently *not* fulfilled for the token signature scheme, but can be achieved using dp-ABCs. The reason is the same as for **Auth-SReq-4**; that the token is partially signed in plain by each of the partial IdPs but not in the case of dp-ABCs.

Auth-SReq-6 *optional*: This requirement states that the adversary should not be able to link the identity of a user across different SPs. Currently this can be fulfilled, depending on the SP policy. If the policy does not contain identifying, such as user ID or exact timestamps, attributes, the requirement can be met, otherwise it can not.

Next we discuss the fulfilment of the account management requirements as defined in Section 2.4, under the assumption that at most $n-1$ partial IdPs are compromised.

Man-SReq-1 : This requirement states that the user's messages should not allow for an offline attack on its password. Given that both registration, authentication and management calls only require the user to use its password as input to the dpOPRF, this requirement is fulfilled by the same argument as for **Reg-SReq-1**.

Man-SReq-2 : This requirement states that a user must know the password used through registration to get an access token. This requirement is fulfilled given that management calls first require an authentication as part of the Login step. Afterwards the signing key is used to sign management calls. Hence the reason this is fulfilled is the same reason that **Auth-SReq-2** is fulfilled.

Man-SReq-3 : This requirement states that the IdPs must be able to limit online attacks. This requirement is fulfilled since the password is only used through authentication during Login, and is hence fulfilled since by **Auth-SReq-3**.

Man-SReq-4 : This requirement states that only a user who knows the password during registration or can gain access to the password reset functionality associated with the user is able to access the user's account. This requirement is fulfilled. First part is fulfilled by **Man-SReq-2**. The second part is fulfilled architecturally since

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

if the user has forgotten his/her password then it must use a token sent out-of-band to redo a registration protocol to the account with its associated information.

Then we discuss the fulfilment of the requirements related to multi-factor authentication as defined in Section 2.5, it is assumed that the partial IdP's are not corrupted unless the AP is part of one IdP in which case only that IdP can be compromised.

2FA-SReq-1 : This requirement states that the messages sent in connection with multifactor authentication must not allow for offline attacks on the user's password. As the protocol with the AP should not include the user's password and $2Ftok$ only includes the AP signature, the transmitted data cannot be used to compromise the user's password.

2FA-SReq-2 : This requirement states that even a compromised AP must not make it easier for an adversary to carry out an offline attack on a user's password. Following the argument for **2FA-SReq-1**, a compromised AP will only allow the adversary to bypass the multifactor authentication and is therefore restricted to online attacks.

2FA-SReq-3 : This requirement states that the AP must not know any of the user's private information. As the AP only knows *a* uid (not necessarily the one used by the vIdP), an AP compromise will not reveal any user information.

2FA-SReq-4 : This requirement states that a user can only add a second factor to an account where she knows the account password. As the user must first authenticate with the vIdP, this is achieved with the same argument as for **Man-SReq-2**.

2FA-SReq-5 : This requirement states that a user can only use 2FA for accounts where the password is known. As $2Ftok$ must be signed by the user's private key sk to be valid and sk can only be obtained by knowing the user's password, this requirement is met.

Finally we discuss the fulfilment of the requirements related to refreshing the epoch keys as defined in Section 2.6, under the assumption that at most $n-1$ partial IdPs are compromised.

Refresh-SReq-1 : This requirement states that a fully compromised partial IdP in one epoch, must not allow the adversary to gain an advantage in attacking the system in a later epoch. That is, as

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

long as one server remains uncompromised within every epoch, then the security of the system remains intact. This is achieved through the concrete refresh mechanism which is cryptographic and independent of up to $n-1$ of the current epoch keys [2].

Refresh-SReq-2 : This requirement states that the refresh mechanism should not allow an adversary to launch a denial of service attack. Since refreshing will make the vIdP unavailable for a short while, this can only be done by an authorized person with knowledge of a special key, which can be independent for each vIdP. Hence an adversary must compromise all partial IdPs to be able to issue refresh queries and hence leave the service unavailable.

4.2 Distributed Token Generation

When the IdPs have collaboratively run the password verification protocol and concluded that the password is correct, they engage in a follow-up protocol to jointly generate the access token tok. In D4.1 we propose two variants: distributed (standard) signatures and distributed issuance of privacy-enhancing credentials.

Again, we sketch their high-level ideas and properties in the following and refer to the full details to D4.1.

	Standard Signatures	Privacy Credentials
Security against impersonation attacks	✓	✓
Compatibility with SSO standards	✓	✘
Blind policy signing	(✓)	✓
Full unlinkability	✘	✓
User-side complexity	low	high (full PABC user engine)
Verifier-side complexity	low (standard verification)	high (full PABC verifier engine)

4.2.1 Distributed Signatures

A distributed signature scheme is similar to a regular (non-distributed) signature scheme, with the only exception that the private signing key,

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

sk is now distributed across n servers such that all n servers must perform a partial signature using their individual shares of the signing key. The partial signatures can then be combined to a single, valid signature with an associated single public key.

Concretely this means that we assume we have a trusted dealer constructing a public verification key vk, and n shares of the private signing key, $sk_1, \ldots, sk_n$. A partial signature can then be constructed on a message m using each of the shares of the private signing key $sk_1, \ldots, sk_n$. These partial signatures can then be combined to the true signature which can be verified using the public key vk. The crux of this approach is that *all* partial signatures are required to construct a true, valid signature.

Most standard signature schemes can be modified to allow for distribution. (EC)DSA generally require the signing to be interactive among the servers holding the private key shares [6, 11], whereas RSA signing can be made distributed trivially [3]. In fact the simplest distributed approach to RSA allows the share-holders of the private signing key to simply perform the standard RSA signing on a message using their share, and combining shares is done by simply computing the product. For this reason we have chosen to use RSA for distributed signatures. Concretely what will be implemented is a distributed version of RSA PKCS#1 v. 1.5 [12] such that a combined signature can be verified by any client able to verify regular RSA PKCS#1 v. 1.5 signatures. We do however note that the generation of a distributed private key for RSA without a trusted third party is very complex and requires heavy cryptographic machinery. For this reason we will assume a trusted third party for key generation in this project and leave it as future work to use a secure distributed RSA key generation protocol [3, 7, 1, 4, 5, 10, 9].

4.2.2 Distributed Credentials

The dp-ABC approach addresses the single point of failure problem in the issuance of user credentials that occurs in traditional p-ABCs. Akin to distributed signatures, each server will have a partial secret key it will use to perform a signature. However, the scheme described in D4.1 is a multi-signature scheme. This means that there is no need to rely on a complex protocol (or a trusted dealer) to distribute the different secret keys. Instead, each server will generate its own secret key and the corresponding public key. The public keys of n servers can then be aggregated to obtain a single public key that will be used to verify signatures obtained by combining the signatures of *all* servers.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

Another difference with respect to the other variant is that the user no longer receives signature shares of the final access token from the IdPs, but rather shares of a credential σ containing all her user attributes. Then, the user derives the final access token as a p-ABC presentation token from the freshly received credential, inheriting the unlinkability and minimal disclosure features from p-ABCs. As the user obtains a credential that is not linked to a specific access policy, she can store and use it for deriving access tokens multiple times while maintaining unlinkability. As an access token, the user generates a zero-knowledge signature of knowledge of σ over a message m selectively disclosing some of the attributes in σ. The verifier must then assert that the zero-knowledge proof is valid (using the public key of the vIdP) and check that the revealed information is enough to fulfil the access policy.

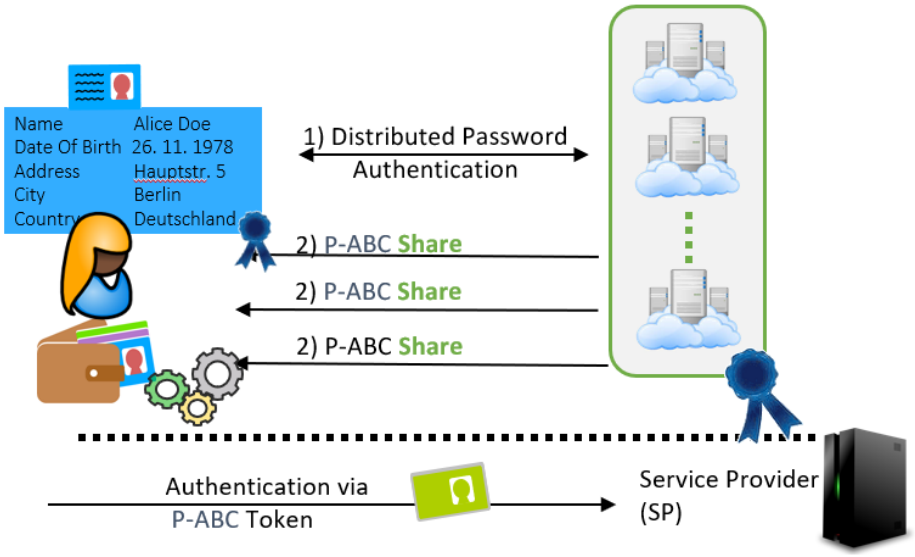


Figure 5: Distributed p-ABC scheme

Thus, this approach solves the problems of the single point of failure (with respect to traditional p-ABCs), minimal disclosure and unlikability. Also, the IdPs (or an adversary) will not learn the access policy nor the service providers that the user is accessing. However, the user-side complexity increases as deriving zero-knowledge proofs for each access is needed, and the generated credentials may have to be stored and managed. Further, the approach loses the compatibility with traditional SSO systems, and the verifier needs a custom implementation to validate the access token presented by users.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

5. Compliance with D3.1 (Requirements)

The architecture defined in this document has been devised with the goals of the OLYMPUS project in mind, like avoiding the IdP becoming a single-point of failure as in traditional SSO systems. More specifically, in Task 3.1 technical and technological requirements derived from the use cases characterized in WP6 were identified and described. These requirements are the baseline for the OLYMPUS intended functionality, and have been one of the main anchors for the development of the components and interfaces described in this document. According to the description made in D3.1, the requirements are divided into four categories: general, security, usability and operational requirements. In the following, we specify which of those requirements are met (or supported) by the architecture blueprint defined in this document, starting from the **general requirements**:

- **Short lived authentication tokens**. While it is not specifically addressed in this document, the expiration of the signatures used for instantiating the different components can be managed in different ways (e.g., adding the expiration time as an extra element for signing), so this is directly supported by the architecture.
- **Unlinkability across SP and sessions**. This can be achieved depending on the concrete access policy and token produced as addressed in *Auth-SReq-6*.
- **User side hardware environment**. It is not necessary to use specific hardware, such as TPUs, in order to make use of the OLYMPUS solution. However, if the user desires offline authentication through the credential approach, secure storage (not necessarily hardware-based) of the credential is needed.
- **Integrable with existing IdM technologies**. As described in Section 3.4, the distributed signatures approach produces tokens that can be constructed to follow the OpenID Connect or other standard formats allowing for standard verification. However, the credential-based solution is not compatible with them and requires custom implementation for verification.
- **Data-minimization**. The *Auth* interface enables data-minimization. It can be achieved by both token generation approaches, distributed signing (signing the specific information) and dp-ABCs (by design).
- **Support of anonymous authentication** Support depends on

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

the SP policy requirements. Assuming only non-identifying attributes are requested/revealed, the user will remain anonymous towards the SP.

- **Personal data processing following the GDPR regulation**. Tackled by the *Account Management* interfaces.
- **Transparency**. The *Retrieve* (plus *Login*) interface allows the user to retrieve all information stored about her.
- **No brute-forcible security critical data stored on IdPs**. This is ensured by Reg-SReq-2 of the *Register* interface.
- **User Data Deletion/Retention:** User data is sent to the vIdP as the final step in each protocol, hence any failure will happen before data leaves the client.

With respect to *security requirements*, we have:

- **No Impersonation by IdPs**. This is ensured by Auth-SReq-2.
- **Hiding SPs from IdP**. Currently support depends on the SP policy requirements, e.g. this is not supported if OIDC must be supported. Contemplated in the *Auth* interface with its generality for privacy-preserving authentication (optional *Auth-SReq-4* and *Auth-SReq-5*).
- **Authentication**. This is ensured by the assumptions on authenticated channels mentioned in Section 2.
- **Data integrity**. Data integrity during transport is assured by channel assumptions mentioned in Section 2. The architecture enables data integrity at rest too, but whether it is ensured will depend on storage policies.
- **Confidentiality**. Confidentiality during transport is assured by channel assumptions mentioned in section 2. The architecture enables confidentiality at rest too, but whether it is ensured will depend on storage policies.
- **Availability**. If any of the vIdPs of our distributed approach is down the level of security remains (per all security requirements that depend on at most $n - 1$ IdPs being compromised), though no new registrations, logins or token generations can take place, so the service is mostly unavailable (however, if a malicious IdP denies service, it can be detected and fixed through the refreshing protocol).

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

- **Access control**. Only users who have successfully authenticated with the distributed protocol are able to access sensitive data and generate access tokens for services (Auth-SReq-2). With respect to access to manage the IdPs themselves, that will depend on specific deployment.
- **Replay**: This is supported by the architecture. Specifically, the components instantiations described in Section 4 are protected against replay attacks by using nonces for both authentication and token generation (more details in Section 4 and D4.1).
- **Token standards**. The mechanism used to log on to OLYMPUS is tamper proof per the *Register*, *Auth* and *Login* interfaces (see also DPV instantiation in Section 4). The authentication tokens generated by the distributed signature scheme uses standard RSA signatures and the dp-ABCs also follow security standards like protection against replay attacks and security against impersonation.
- **User cannot falsify or change authentication tokens.**: The tokens used in both approaches are verified using the public key of the vIdP with the *Verify* interface. Per *Auth-Sreq-2*, only a user that knows the correct password can obtain a valid access token by successfully authenticating against the vIdP. Also, both instantiations for token generation are unforgeable (details in D4.1).
- **Proactive security**. Addressed in Section 2.6, currently assured for DPV and DTG but not accomplished for the distributed credential approach instantiation (though it is more of a cryptography specific problem, as the architecture presented in the document supports proactive security)

Finally, the **usability and operational requirements** covered by the OLYMPUS project are mostly an application specific problem, though there is some discussion that can be done for the interfaces defined in this document. For example, as the approach consists in using password-based and two-factor authentication, which are really common methods, the **Effectiveness** requirement will be simple to fulfil. Also, the interfaces defined in this document do not define too complicate interactions and the cryptographic primitives are not too heavy, so requirements like **Efficiency**, **Satisfaction** and **Mobile support** seem feasible. In fact the use case demonstrators both aim to document this. With respect to the **Added value for stakeholders involved**, the proposed solution puts a lot of emphasis on guaranteeing user security (e.g., even with $n - 1$ servers compromised the user is pro-

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

tected against identity theft) and privacy (enabling minimal disclosure and full unlinkability). Lastly, the components and interfaces allow flexibility that enables **Interoperability**. For instance, the two approaches proposed for distributed token generation have different characteristics that make them better for specific scenarios (e.g., using credentials for offline authentication processes).

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725

References

[1] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. pages 417–432, 2002.

[2] Carsten Baum, Tore Kasper Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. PESTO: proactively secure distributed single sign-on, or how to trust a hacked server. *IACR Cryptology ePrint Archive*, 2019:1470, 2019.

[3] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001.

[4] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, robust and constant-round distributed RSA key generation. pages 183–200, 2010.

[5] Ivan Damgård, Gert Læssøe Mikkelsen, and Tue Skeltved. On the security of distributed multiprime RSA. pages 18–33, 2015.

[6] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1051–1066. IEEE, 2019.

[7] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. pages 663–672, 1998.

[8] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 331–361. Springer, 2018.

[9] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. pages 331–361, 2018.

[10] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient RSA key generation and threshold Paillier in the two-party setting. pages 313–331, 2012.

[11] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. pages 1837–1854, 2018.

[12] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, RFC Editor, November 2016.

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 786725